НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр КОВАЛЬ

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення веб-технологій та мобільних пристроїв»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Веб-система для інтерактивної роботи з 3D об'єктами»

Виконав:
студент IV курсу, групи ТІ-62
Стеценко Денис Олегович _____

Керівник:
старший викладач
Мірошніченко Іван Володимирович _____

Консультант з розділу 4
к.т.н., доцент,
Шалденко Олексій Вікторович _____

Рецензент:
_____
_____ _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. КОВАЛЬ
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

Стеценко Денису Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-система для інтерактивної роботи з 3D об’єктами

керівник роботи ст. викл. Мірошніченко Іван Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від “25” травня 2020р. № 1168-с

2. Строк подання студентом роботи “12” червня 2020р.

3. Вихідні дані до роботи мова програмування JavaScript, фреймворки Vue.js та Flutter (для веб та мобільної системи відповідно), бібліотека Three.js, методологія JAMStack.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати предметну область 3D моделювання, спроектувати систему за допомогою мови моделювання UML, створити адаптивний дизайн та брендинг програмного продукту, розробити програмне забезпечення веб-системи для інтерактивної роботи із 3D об’єктами використовуючи стандартні методи розробки веб-застосунків та мобільної системи використовуючи засоби для написання кросплатформних застосунків, використати сучасні методології для розгортання веб-системи без виділеного серверу.

5. Перелік ілюстративного матеріалу

Актуальність, Проблеми, Проблеми (перевантаженість інтерфейсу), Постановка задачі, Діаграма прецедентів (спрощена), Діаграма класів (спрощена), Діаграма розгортання (моб. застосунок), Використані технології, Головна сторінка веб-додатку, Інтеракція з моделлю, Навчальний посібник, Мобільний додаток, Переваги створеного продукту, Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	к.т.н., доц. Шалденко О. В.	20.05.20	03.06.20

7. Дата видачі завдання "_11_"_____жовтня_____ 2019_ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	15.10.2019	
2.	Вивчення та аналіз задачі	13.04.2020	
3.	Розробка архітектури та загальної структури системи	20.04.2020	
4.	Розробка структур окремих підсистем	27.04.2020	
5.	Програмна реалізація системи	11.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020	
7.	Захист програмного продукту	26.05.2020	
8.	Передзахист	10.06.2020	
9.	Захист	17.06.2020	

Студент _____ _Стеценко Д. О._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _Мірошніченко І. В._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дана дипломна робота присвячена розробці веб-системи для інтерактивної роботи з 3D об'єктами.

Метою роботи є створення програмного забезпечення доступної, єдиної платформи для швидкого і зручного перегляду 3D об’єктів, інформації про них, взаємодії з оточенням, анімаціями, управлінням та інше.

Для досягнення мети були вирішені наступні задачі:

1. Проаналізовано предметну область 3D моделювання та графічного дизайну, досліджений стан 3D типів файлів на сьогоднішній день.

2. Спроектована система за допомогою мови моделювання UML.

3. Створений адаптивний дизайн та брендинг програмного продукту.

4. Розроблено програмне забезпечення веб-системи для інтерактивної роботи із 3D об’єктами використовуючи стандартні методи розробки веб-застосунків, а також JavaScript фреймворк Vue.js та бібліотеку Three.js для завантаження та інтеракції із 3D моделями та оточенням. Для розробки мобільного застосунку був використаний фреймворк для написання кросплатформних застосунків Flutter.

5. Використано сучасну методологію JAMStack (JavaScript, API, Markup) для розгортання веб-системи без виділеного серверу.

Ключові слова: 3D, моделювання, графічний дизайн, веб-переглядач, інтерактивність.

Обсяг звіту становить 55 сторінок, міститься 23 ілюстрації, 3 додатки. Загалом опрацьовано 22 джерела.

ABSTRACT

This thesis is devoted to the development of a web-based system for interactive work with 3D objects.

The goal is to create an accessible, united platform for swift and convenient viewing of 3D objects, information about them, interaction with the environment, animations, controls and more.

To achieve this goal, the following tasks were solved:

1. The subject area of 3D modeling and graphic design and the current state of 3D file types were analyzed.

2. The system was designed via UML modeling language.

3. Adaptive design and branding of the software were created.

4. Web system software for interaction with 3D objects using standard web application development methods, as well as Vue.js JavaScript framework and Three.js library for downloading and interacting with 3D models and environments was developed. To develop a cross-platform mobile application Flutter framework was used.

5. The modern JAMStack methodology (JavaScript, API, Markup) was used to deploy a web system without a dedicated server.

Keywords: 3D, modeling, graphic design, web-viewer, interaction.

The report volume is 55 pages, it contains 23 illustrations, 3 appendices. 22 sources were processed in total.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	–	Application Programming Interface
BEM	–	Block Element Modifier
DOM	–	Document Object Model
GLTF	–	GL Transmission Format
GUI	–	Graphical User Interface
JAMStack	–	JavaScript, API & Markup
JSON	–	JavaScript Object Notation
SRS	–	Software Requirements Specification
UI	–	User Interface
UML	–	Unified Modeling Language

ВСТУП

3D моделювання — це процес прийняття певної форми та її трансформації у завершену 3D сітку. Примітивами називають прості предмети — від однієї точки (вершини), двовимірної лінії (ребра), кривої (сплайну), до тривимірних об’єктів (граней чи багатокутників) [1]. Використовуючи особливості такого програмного забезпечення як 3D переглядач (3D Viewer) можна маніпулювати комплексами примітивів для перегляданння 3D моделі, регулювати освітлення, переглядати анімації та багато іншого.

Напрям 3D моделювання — один із найдинамічніших напрямів IT-сьогодення, який використовується для створення 3D моделей для різних сфер діяльності. Використовувані в таких галузях, як анімація [2], ігри, архітектура, виробництво, ітерація виробів та промисловий дизайн, 3D моделі є найважливішими компонентами цифрового виробництва.

Як один із прикладів, будівельні фахівці постійно придумують нові ідеї інноваційних дизайнів, і корисно мати програмне забезпечення, яке дозволяє швидко перетворити ці ідеї в тривимірні зображення [3]. Програмне забезпечення для 3D моделювання дозволяє будівельним професіоналам перетворювати свої конструкції та креслення в реальні моделі та дозволяє художникам чітко уявляти розміри своїх будівельних конструкцій до завершення будівництва [4].

Програмне забезпечення для 3D моделювання дозволяє створювати, розмежувати та вдосконалювати тривимірні фігури в будь-який об’єкт, який ви можете собі уявити [1]. Однак галузь змінюється настільки швидко, що визначити, яка програма 3D моделювання є найкращою для певних 3D потреб, може бути досить складно, оскільки протягом останніх років галузь не лише зросла в десятки разів, а й також постійно впроваджує нові поняття та способи їх здійснення [5] [6].

На даний момент, програмні продукти, створені суто для переглядання інформації про 3D модель, є обмеженими та здебільшого розробляються компаніями як додаток до об’ємних та важких систем, пов’язаних із 3D моделюванням. Гострою проблемою також слід вважати пропрієтарність форматів моделей та складність конвертування їх між собою, яка позначається на зменшенні ефективності роботи над 3D об’єктами [7]. Таким чином, не існує єдиної платформи для зручного перегляду 3D моделей та інформації про них.

Заради вирішення описаних проблем реалізована веб-система, що орієнтується на доступність, легкість у використанні, швидку взаємодію та призначається виключно задля мети інтеракції з 3D моделлю. Створений продукт має досить велику низку сфер застосування, насамперед — анімаційні фільми, візуальні ефекти, графічний дизайн, 3D друковані моделі, кінофільми, інтерактивні 3D застосунки та ігрова індустрія, а завдяки своїй веб-архітектурі, застосунок є легко-доступним та не вимогливим з точки зору апаратного забезпечення.

Очікується, що продукт буде використовуватися у багатьох сферах, таких як 3D моделювання, ігрова індустрія, графічний дизайн та інші. Створена також додаткова підтримка для мобільних пристроїв як iOS так і Android в якості створення мобільного додатку як більш зручної нативної версії веб-системи.

Зміст розділів пояснювальної записки є наступним:

— Перший розділ описує постановку задачі створення програмного комплексу, її мету, проблеми, які вирішуються програмним забезпеченням та схему взаємодії програмних модулів.

— Другий розділ проводить огляд вже існуючих програмних рішень та автоматизованих систем на ринку.

— Третій розділ надає інформацію про задачі та цілі модулей комплексної системи. Так як реалізований продукт складається з двох частин — основної веб-системи та додаткового мобільного додатку — виконується порівняльна характеристика, обґрунтування та розбір методів вибору прикладних і системних

засобів програмування, середовищ проектування, методологій розробки програмного забезпечення тощо.

— Четвертий розділ описує безпосередню програмну реалізацію системи та є документальним супроводом розробленого програмного забезпечення. До нього входять UML діаграми класів, прецедентів, послідовностей та розгортання (для мобільного додатку), описуються особливості реалізованих підсистем та система взаємодії модулей між собою.

— У п’ятому розділі пояснюється методика роботи користувачів з програмною системою. У цьому розділі розібрані сценарії роботи користувача та послідовності стимулу / відповіді, надані візуальні приклади взаємодії користувача із програмним комплексом.

1. ПОСТАНОВКА ЗАДАЧІ ПОБУДОВИ КОМПЛЕКСНОЇ СИСТЕМИ ДЛЯ ІНТЕРАКЦІЇ З 3D МОДЕЛЯМИ

Метою роботи є аналіз поточного стану сфери 3D моделювання та створення інтерактивної веб-системи, яка дозволяє користувачам взаємодіяти з 3D об'єктами певних підтримуваних форматів. Проект представляє собою доступну, єдину платформу для швидкого та зручного перегляду 3D моделей і інформації про них, взаємодії з оточенням, анімаціями та контролями управління. Паралельно до основної веб-системи також створений додатковий кросплатформений мобільний додаток, у якому проаналізовано та прокласифіковано найбільш використовувані типи 3D файлів на сьогоднішній день.

В ході роботи було визначено основні завдання роботи:

— проаналізувати поточний стан програмного забезпечення для сфер 3D моделювання та 3D інтеракції;

— створити програмний комплекс, що складається із інтерактивної веб-системи для роботи із 3D моделями та кросплатформеного мобільного застосунку для аналізу та класифікації найпоширеніших типів 3D файлів сьогодення;

Під час декомпозиції основних завдань були визначені наступні підзадачі:

— проаналізувати поточний стан веб-розробки та визначитися зі стеком технологій та методологій, необхідним для створення програмного комплексу;

— проаналізувати поточний стан програмного забезпечення для сфер 3D моделювання та 3D інтеракції;

— створити початковий брендинг та спроектувати дизайн системи;

— проаналізувати поточний стан типів файлів 3D моделей, провести їх класифікацію та визначитися з їх підтримкою;

— спроектувати архітектуру веб-системи та мобільного застосунку використовуючи мову проектування UML;

— створити кросплатформений мобільний застосунок для аналізу та класифікації найпоширеніших типів 3D файлів сьогодення;

— створити модуль завантаження 3D об’єкту в веб-браузер;

— створити модуль оточення для взаємодії із завантаженим об’єктом;

— проаналізувати необхідні елементи контролю 3D об’єкта та його оточення;

— створити графічний інтерфейс користувача (GUI — Graphical User Interface) для контролюючої взаємодії із моделлю;

— провести тестування роботи системи використовуючи комплексні та великі за об’ємом 3D моделі для виявлення якості виконання системи під навантаженням;

— провести подальше тестування й налагодження створеної програмної системі та загальні e2e (end-to-end) тести для перевірки якості роботи системи;

— провести розгортання (deploy) створеної веб-системи на хостинг.

Вимоги до розроблюваного програмного комплексу:

— комплекс має складатися з двох підсистем:

- основна веб-система для інтеракції з 3D об'єктами;
- додатковий нативний мобільний застосунок як розширена альтернатива до веб-системи для смартфонів;

— веб-система має дотримуватися функціональних та нефункціональних вимог, описаних у доданому документі SRS (Software Requirements Specification), зокрема повинна мати підтримку більшості браузерів та дотримуватися спроектованого адаптивного дизайну;

— веб-система повинна дозволяти повну навігацію і вибір інтерактивних дій із використанням тільки миші, комбінацій миші і клавіатури, а також сенсорних жестів (для мобільного додатку або мобільної версії веб-сайту);

— веб-система має бути розгорнута на веб-хостинг та слідувати методології JAMStack (JavaScript, API & Markup).

Вхідною інформацією до веб-системи виступають наступні дані:

— інформація про 3D модель, що завантажитиметься у веб-браузер:

- розширення (формат) 3D файлу;

- розташування об’єкту у файловій системі пристрою користувача;
- безпосередньо буфер інформації 3D моделі;

— інформація про користувача та локальні дані, що надає клієнт браузеру;

Вихідною інформацією є наступні дані:

— інформація про завантажену модель:

- кількість геометричних примітивів;
- наявність та кількість анімацій моделі;
- наявність та кількість сцен оточення моделі;
- наявність скелету моделі для використання у фізичних двигунах;

— інтерактивний GUI для контролю світла, анімацій, оточення тощо;

— контролі управління для перегляду завантаженої моделі та оточення.

Документація для користувача розподілена на дві платформи:

— документація на офіціальній сторінці системи контролю версій GitHub;

— вбудований безпосередньо в системі навчальний посібник із використання.

Розроблений продукт можна використовувати як швидку та єдину платформу для перегляду 3D моделей різних форматів. Завдяки своїй адаптивності, веб-додаток має можливість бути вбудованим у різні платформи 3D індустрії, маючи перевагу над пропрієтарними, водночас менш специфічними та гнучкими системами. Таким чином, процес перегляду та інтеракції з майже будь-якою 3D моделлю стає простішим та доступнішим. За рахунок побудови його на основі веб-технологій реалізується універсальність та зручність використання на будь-якому пристрої без перенесення на цільову операційну систему (браузер та його віртуальна машина виступають як цільова універсальна операційна система та комп'ютер).

У даному документі описаний повний цикл розробки програмного забезпечення для роботи комплексу, який починається з аргументації обраного програмного стеку, методології розробки і розгортання веб-системи та закінчується детальним описом сценаріїв інтеракції кінцевого користувача із створеним програмним комплексом.

2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ДЛЯ ІНТЕРАКТИВНОЇ РОБОТИ З 3D МОДЕЛЯМИ

Нещодавно для створення 3D моделей було необхідним дороге програмне забезпечення, що працює на спеціалізованих операційних системах та спеціалізованому обладнання, але ці часи вже минули. Наразі, більшість комп'ютерів мають достатньо графічної продуктивності для моделювання досить складних конструкцій, як, наприклад, інтер'єр будівлі або об'єкт з 3D друком. І там, де колись для заявок потрібна спеціальна підготовка та контракти на обслуговування, зараз існує безліч безкоштовних чи недорогих інструментів, які допоможуть легко та ефективно спроектувати складні форми та механізми [8].

За допомогою цієї еволюції програмне забезпечення для 3D моделювання урізноманітнюється у спеціалізовані інструменти для створення прототипів, візуальних ефектів, моделювання та інших конструктивних функцій, які корисні для багатьох галузей [9], включаючи архітектуру, дизайн інтер'єру, ігрова індустрія, ландшафтний дизайн, а також макетне програмне забезпечення для продукту ідеї. Незалежно від того, який тип 3D виходу вам потрібен, існує програма, яка ідеально підходить до завдання, і шлях до розвитку навичок, необхідних для ефективного використання.

На даний момент вже створено досить багато програмного забезпечення, за допомогою якого користувач може редагувати комплексні моделі, вбудовувати ці моделі у додаткові плагіни для роботи у спеціалізованій сфері (наприклад імпортування створеної моделі із анімаціями у Adobe After Effects для створення кінематографічних послідовностей у фільмах) [2] [10]. Однак гострою проблемою в такому випадку стає те, що немає специфічної платформи створеної суто для швидкого та зручного перегляду вже створеної моделі, адже всі створені, досить

коштовні продукти є вкрай важкими та переповненими функціоналом, яким більшість користувачів мабуть ніколи не скористається [11].

З метою вирішення цією проблеми було запропоновано розробити програмний комплекс спеціалізованої, єдиної платформи для поверхневої інтеракції з 3D моделлю. Така система орієнтується на доступність та зручність для користувачів, має широку перспективу у використанні, оскільки може бути вбудованою у більш комплексну систему для повноцінної роботи створення, редагування та перегляду 3D об’єктів і має низку сфер застосування — візуальні ефекти, графічний та промисловий дизайн, 3D друковані моделі [12] [13] [14], кінофільми, інтерактивні 3D застосунки та багато інших. А з точки зору вимогливості та доступності, продукт є досяжним і легким для розуміння сервісом завдяки своїм веб та мобільній архітектурам (для веб-системи та нативного додатку відповідно).

2.1 Аналіз існуючих програмних засобів

Внаслідок пошуку вже існуючих аналогічних програмних систем та рішень було детально проаналізовано наступні реалізовані програмні продукти:

— Microsoft 3D Viewer — Новий тривимірний переглядач, що впроваджується для всіх користувачів операційної системи (ОС) Windows за допомогою магазину Microsoft та заміняє раніше створений додаток Mixed Reality Viewer. Сам додаток було перероблено з нуля, із інтерфейсом, який має набагато більше сенсу при використанні миші та клавіатури. Додаток підтримує велику кількість найбільш поширених розширень 3D файлів і також підтримує змішану реальність за допомогою камер пристроїв, що дозволяє розміщувати віртуальні об'єкти в реальному світі, як це робив попередній додаток змішаної реальності. Також є прямий доступ до програми Paint 3D через рядок меню, який автоматично перетягне 3D модель у додаток Paint 3D для редагування. Недоліком цього продукту є строга

належність до ОС Windows як десктопного програмного забезпечення, що робить систему неможливою для використання іншими ОС;

— Sketchfab Viewer — переглядач на основі технологій WebGL та WebVR платформи для публікації, обміну, відкриття, купівлі та продажу вмісту 3D, VR та AR Sketchfab. Він дозволяє користувачам відображати 3D моделі в Інтернеті, переглядати їх у будь-якому мобільному браузері, настільному браузері або гарнітурі Virtual Reality. API переглядача дозволяє створювати веб-додатки над 3D переглядачем Sketchfab, надає функції для запуску, зупинки глядача, переміщення камери, зйомки екрана тощо. Єдиними недоліками даного переглядача є його тісна прив'язаність до об'ємної платформи та велика кількість платного функціоналу;

— BabylonJS — один із найпотужніших та простих у використанні двигунів веб-рендерінгу, який орієнтується на повну відкритість та безкоштовність. Babylon.js 4.1 включає безліч оптимізацій продуктивності, продовжуючи лінійку високопродуктивного двигуна. Редактор має велику кількість під-систем, які активно розвиваються внаслідок відкритого коду проекту. Також, значним плюсом є зручний та зрозумілий у використанні інтерфейс. Великим мінусом для цього проекту є підтримка лише одного формату 3D файлів — GLTF (GL Transmission Format), який визначає розширюваний, загальний формат публікації для інструментів та служб 3D вмісту, впорядковує авторські робочі процеси та дозволяє оперативно використовувати вміст у всій галузі [15] [16];

— 3D Viewer Online — веб-переглядач 3D моделей, який підтримує досить велику кількість поширених 3D типів файлів та має зручний та інтуітивний графічний інтерфейс користувача. Важливим недоліком даної системи є великі обмеження у розмірі завантажених моделей (навіть у випадку придбання бізнес-плану продукту). Слід зазначити, 3D Viewer Online значно обмежує функції переглядача при використанні безкоштовної версії, таких як: реалістичні відображення, задній фон та оточення моделі, використання кольорів, налаштування макету панелі інструментів та можливість обертання завантаженої 3D моделі. Отже, як і в попередньому прикладі, великим недоліком приведеного програмного

комплеку є щільна прив’язаність до об’ємної платформи та велика кількість недоступного або ж платного функціоналу.

Таким чином, можна дійти висновку, що на даний момент проаналізовані системи є досить складними у використанні, не мають всебічної доступності, або реалізують поставлену задачу не в повному обсязі. Ще одним недоліком цих систем є їх громіздкість, внаслідок якої для початкового користувача ці реалізовані продукти можуть здатися функціонально переповненими. Отже розробка системи, яка заснована на адаптивних веб-технологіях, є пріоритетною задачею для вирішення проблеми доступності для найбільш широкого кругу користувачів.

3. ЗАСОБИ РОЗРОБКИ ДЛЯ РЕАЛІЗАЦІЇ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Архітектура програмного комплексу

JAMStack (JavaScript, API, Markup) — сучасна архітектура веб-розробки на основі клієнтського JavaScript, багаторазового використання API та попередньо встановленої розмітки [17]. Створена таким чином архітектура розмежує функціональну відповідальність на три основні чинники:

— Мова JavaScript обробляє усі динамічні функціональності;

— Абстраговані API (Application Program Interface) використовуються для операцій зі серверною стороною, такими як сторонні сервіси або спеціальні функції;

— Розмітка (Markup) веб-сайтів подається як статичні HTML (Hypertext Markup Language) файли, які при цьому можна генерувати з вихідних файлів використовуючи статичні генератори сайтів.

Завдяки методології JAMStack, створені веб-системи отримуються наступні переваги:

— Більш швидке виконання внаслідок заздалегідь розміченої розмітки та активів, що передаються через CDN (Content Delivery Network);

— Покращена безпека, так як більше не використовується прямий зв’язок до серверів чи баз даних;

— Менші витрати на зберігання веб-систему. Хостинг статичний файлів розмітки на даний момент є вкрай дешевим або навіть безкоштовним;

— Швидка і більш цілеспрямована розробка. Розробники front-end мають можливість повністю зосереджуватися на своїй задачі, не прив’язуючись при цьому до монолітної архітектури;

— Краща масштабованість у разі поширення веб-системи внаслідок компенсації CDN.

Сайтами JAMstack також можна керувати за допомогою системи управління вмістом, як правило, вони відомі як CMS (Content Management System) без головного управління. Після внесення змін у CMS буде запущено нове складання вашого веб-сайту та його розгортання як статичних активів (static assets) [17].

Абстрактний робочий процес JAMStack зображений на рисунку 3.1.

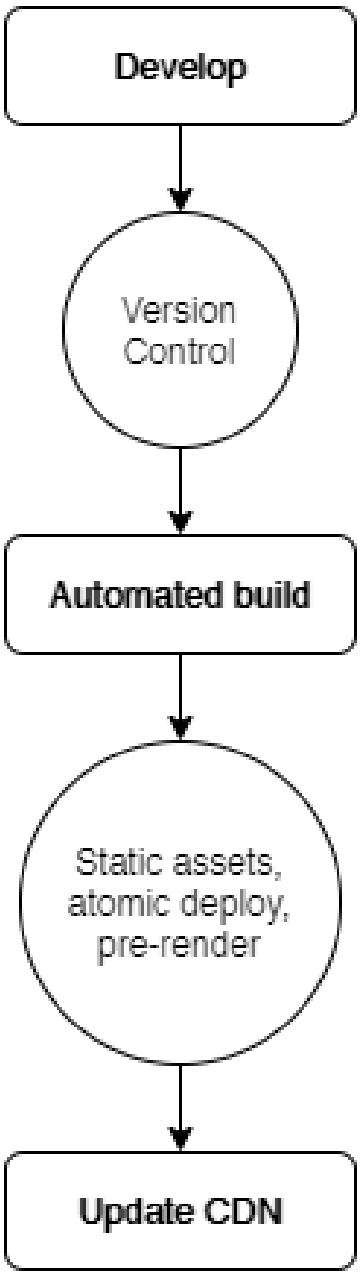

Рисунок 3.1 — Абстрактний процес JAMStack

3.2 Поточний стан веб-розробки

Мова розмітки гіпертексту (HTML — Hypertext Markup Language) має важливе значення для кожного веб-сайту, який ви реалізуєте. За допомогою HTML ви визначаєте та структуруєте вміст веб-сайту за допомогою простого синтаксису розмітки. Каскадні таблиці стилів (CSS) — це простий спосіб додавання стилів на веб-сайти (наприклад, додавання шрифтів, кольорів, макета тощо). Під час розробки була використана фронт-енд методологія розробки BEM (Block Element Modifier).

Мова програмування JavaScript є найважливішим складовим елементом для додавання динамічних функцій на ваш веб-сайт, це реалізація основної специфікації ECMAScript. Вивчення JavaScript з нуля є важливою віхою для початку вашої кар’єри в галузі веб-розробок. Для багатьох фреймворків на front-end (Angular, React, Vue.js) та на стороні back-end (Node.js) знання JavaScript є обов'язковою умовою.

Із поширенням фреймворку Angular, TypeScript також знайшов широке визнання. TypeScript є доповненням до JavaScript яке можливо використовувати водночас із серверною платформою Node.js (розглядається далі), тому завжди використовується разом із JavaScript, щоб увімкнути додаткові функції, прикладами яких є:

— статична типізація;

— підтримка повноцінної класової моделі;

— підключення модулів;

Фреймворки — як для розробки front-end, так і для back-end — спрощують розробку та надають можливість швидше розвивати загальну функціональність. Однак перед тим, як почати використовувати фреймворк, ви завжди повинні намагатися глибше зрозуміти базовий стек технологій (наприклад, JavaScript). Це допоможе вам засвоїти рамки та зрозуміти концепцію.

Найбільш поширені на сьогоднішній день front-end фреймворки у веб-розробці є наступними:

— Vue.js — надзвичайно простий у вивченні, ідеальний вибір для проектів менших-середніх масштабів. Крива навчання цього фреймворку надзвичайно крута, тому можна швидко побачити чудові результати за короткий час після вивчення основ фреймворку. Цей фреймворк був обраний для розробки даної веб-системи, використовуючи також схему управління даними і сервісами Vuex, що значно спрощують розробку проектів із глобальним сховищем даних і роблять код більш передбачуваним та простішим для подальшого налагодження;

— React — один з найпопулярніших front-end фреймворків сьогодення. React повністю базується на компонентах, при чому можна створити інкапсульовані компоненти, які керують власним станом, а потім скласти їх для створення складних інтерфейсів. Оскільки логіка компонентів написана в JavaScript замість шаблонів, ви можете легко передавати багаті дані через додаток і не підтримувати стан DOM;

— Angular є основою для побудови клієнтських додатків у HTML та JavaScript або TypeScript. Цей фреймворк поєднує в собі декларативні шаблони, введення залежностей, інструменти end-to-end тестування та заздалегідь інтегровані найкращі практики для вирішення проблем розвитку. Angular є найкращим front-end фреймворком для написання комплексних та важких веб-системи, тому його використання для легковісних, спеціалізованих задач не є оптимальним. До того ж, крива навчання для цього фреймворку є більш похилою і його освоєння потребує значне більше часу ніж для React або Vue.

Існує також багато варіантів back-end фреймворків, які запускаються на сервері і динамічно формуються HTML, JSON (JavaScript Object Notation), тощо для веб-сайту, залежно від URL-адреси, яку відвідує користувач. Одним із найпоширеніших рішень в цій сфері є Node.js. Це програмний продукт є надзвичайно потужним, особливо якщо він використовується разом із проміжним програмним забезпеченням (middleware) Express.js.

3.3 Засоби 3D моделювання, Three.js

Three.js — легковісна бібліотека мови JavaScript, що використовується для створення та відображення анімованої комп’ютерної 3D графіки при розробці веб-додатків. Three.js скрипти можуть використовуватися спільно з елементом HTML5 CANVAS, SVG або WebGL (Web Graphics Library). Three.js дає можливість створення 3D графіки із використанням прискорення графічної відеокарти без підключення пропрієтарних плагінів для браузера клієнта. Варто зазначити, що дана бібліотека має підтримку усіх веб-браузерів які використовуються на сьогоднішній день.

Three.js часто плутають з WebGL, оскільки частіше за все, але не завжди, Three.js використовує WebGL для малювання 3D. WebGL — це система дуже низького рівня, яка малює лише точки, лінії та трикутники [18]. Для того, щоб досягти великих результатів з WebGL, як правило, потрібно досить багато коду. Саме для цього існує Three.js, що обробляє такі речі, як оточення (сцени), світло, тіні, матеріали, текстури, 3D математику, тобто всі речі, які раніше доводилося писати самостійно при беспосередньому використанні WebGL.

Основними можливостями бібліотеки Three.js є:

— Додавання або видалення об'єктів в режимі реального часу використовуючи сцени та оточення;

— Три види камер для перегляду сцен — ортографічна, перспективна та стерео-камера;

— Великі можливості у сфері анімацій, зчитування каркасів моделі, швидка та зворотня кінематика;

— Деталізоване управління джерелами світла, велика кількість видів світла — навколишнє, спрямоване, півкульне, точкове, прожекторне та світло прямокутної області;

— Величезна кількість 3D об’єктів, повноцінна 3D геометрія;

— Завантажувачі даних (loaders), що підтримують велику кількість різних розширень 3D файлів, можливість завантаження об’єкту як у двійковому форматі або зображення [19][20], так і у вигляді JSON-інформації;

— Детальна документація API бібліотеки, публічний форум, велике співтовариство та величезне сховище прикладів роботи із кожним компонентом бібліотеки.

Структура додатку Three.js зображена на рисунку 3.2 [21].

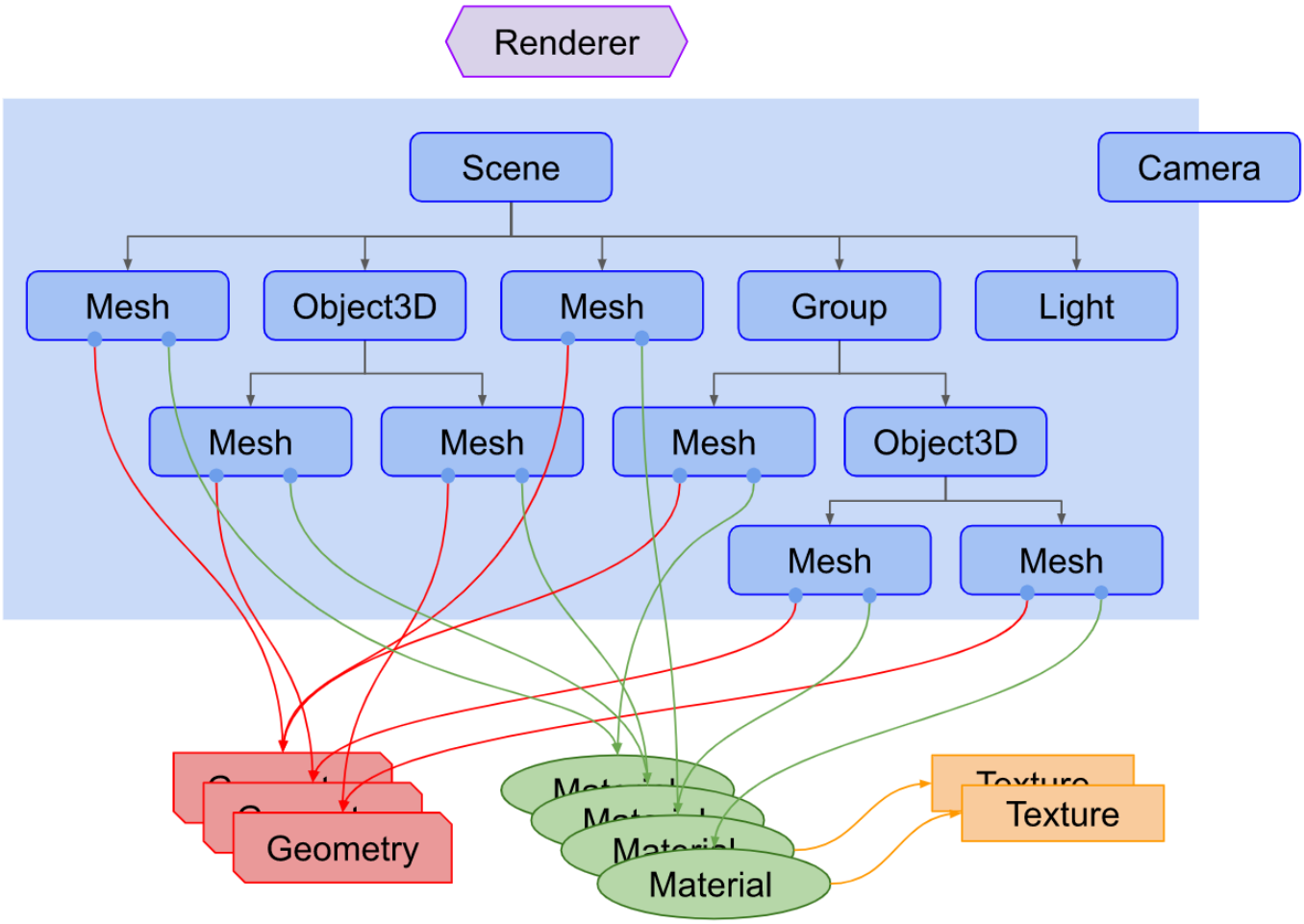

Рисунок 3.2 — Структура додатку Three.js

3.4 Засоби мобільної розробки, Flutter

На даний момент існує два популярні методи мобільної розробки — нативна та кросплатформна, які ідеально підходять для Android та iOS.

Нативна розробка — це розробка додатків, які мають кодову базу, створену для роботи на певній операційній системі — iOS або Android (або будь-якій іншій). Наприклад, Instagram, не дивлячись на те, що на обох операційних системах (далі ОС) виглядає досить схоже, представлений двома власними програмами з окремими кодовими базами. Програми написані окремо, щоб націлити користувачів обох ОС.

Переваги нативної розробки:

— Нативні програми є найшвидшими адже написані на «рідній» для мобільної ОС мові;

— Максимальні переваги до іншого нативного функціоналу та простий доступ до push-сповіщень;

— Можливість доступу до власних функцій телефону

З нативним методом розробники можуть мати доступ до набору функцій мобільного телефону, наприклад, GPS, камери, динаміка тощо. Крім цього, є можливість легко використовувати push-сповіщення, які надають власникам додатків можливість зробити своїх користувачів більш продуктивними. Крім того, нативні програми є високопродуктивними та швидкими.

Незважаючи на всі переваги, які ви отримуєте від власних додатків, цей тип розробок має ряд недоліків:

— Надмірні витрати на бюджет розробки;

— Ресурсоємна розробка;

— Повільна доставка та запуск програмних додатків;

— Складність у підтримці.

Нативні програми iOS не можна запускати на Android та навпаки. Ось чому, щоб максимально використати свій продукт, потрібно мати дві окремі кодові бази.

Це означає, що витрачається великий бюджет внаслідок найняття двох команд. Розробка та доставка таких додатків займає багато часу, а їх підтримка завдає велику кількість клопоту, оскільки вам доведеться виправляти проблеми двох нативних програм, а не однієї.

Кросплатформна мобільна розробка — це метод розробки додатків, який дозволяє програмному продукту працювати на декількох мобільних операційних системах і бути написаним однією мовою програмування одночасно. Коли код програми готовий, він проходить через проміжне програмне забезпечення, яке переводить його в нативну API для iOS або Android.

Переваги такого методу:

— Швидкий розвиток за короткий проміжок часу;

— Економічна ефективність;

— Код багаторазового використання;

— Ідеальна бізнес-можливість залучення великої частки ринку мобільних додатків.

Цей тип мобільної розробки дає шанс охопити найбільшу частку ринку мобільних пристроїв за допомогою програми, що підходить для всіх основних операційних систем. Це означає заощадження грошей, оскільки для запуску програми на кількох ОС потрібна лише одна база коду. Використовуючи кросплатформений метод, розробники отримують прибуткову можливість створити додаток, який буде привабливим для Android, iOS та інших користувачів ОС, не витрачаючи велику кількість коштів.

Але і цей метод має низку недоліків на які варто звернути увагу:

— Обмежений доступ до нативних функцій мобільних пристроїв;

— Проблеми із досвідом роботи користувачів;

Кросплатформна розробка надає обмежену можливість набору функцій телефону, оскільки всі пристрої мають різну функціональність. Як результат, ви можете вирішити деякі проблеми щодо користувацького досвіду, оскільки всі пристрої мають власну функціональність та платформи. Таким чином, це робить

мобільний додаток між платформами менш конкурентоспроможним порівняно з рідними програмами.

Flutter — це фреймоврк з відкритим кодом для створення високоякісних мобільних додатків високої продуктивності в мобільних операційних системах — Android та iOS. Він пропонує простий, потужний, ефективний та простий у розумінні SDK для написання мобільного додатку рідною мовою Google, Dart [22].

Випущений корпорацією Google у 2017 році, Flutter створює вбудовані програми, які можуть підтримуватися різними операційними системами, а основною мовою програмування є Dart, що забезпечує підвищення продуктивності до 15%. З оновленням у 2019 році Google додав корисну функцію — отримання підтримки під час інтеграції Flutter у існуючий додаток для iOS або Android, представлений у магазині. Серед прихильників Flutter можна знайти такі компанії, як Abbey Road Studios, Alibaba, BMW, Groupon, Google, Tencent та багато інших.

3.5 Обґрунтування вибору програмної реалізації

Після ретельного аналізу предметної області та дослідження сфер 3D моделювання і розробки веб-застосунків було вирішено розроблювати програмний комплекс використовуючи front-end фреймворк Vue.js та бібліотеку для взаємодії із 3D оточення Three.js мови JavaScript. При цьому, великої уваги надається методології JAMStack та її основоположним принципам для розробки та розгортання програмних продуктів (веб-систем).

Фреймворк Vue.js є об’єктивно найкращим із “трьох гігантів front-end” (React, Angular, Vue) для створення легковісних та швидких веб-застосунків в силу своєї архітектури, пристосованості до методології JAMStack, легкості у вивченні та простій взаємодії із іншими бібліотеками та фреймворками мови JavaScript. Vue використовує класичні веб-технології та створює свої надбудови над ними, що призводить до покращення швидкості виконання створених веб-застосунків.

На відміну від інших монолітних фреймворків, Vue.js розроблений практично з нуля, щоб мати можливість інкрементно адаптуватися до змін веб-розробки. Основна бібліотека є лише шаром перегляду, а не цілковитим фреймворком-екосистемою для розробки, тому з нею легко розпочати інтегрування з іншими бібліотеками або існуючими проектами. З іншого боку, Vue також чудово здатний живити складні додатки для однієї сторінки, коли їх використовують у поєднанні із сучасними бібліотеками та іншими підтримуючими інструментами.

Проаналізувавши сферу 3D графіки у веб-браузерах можна з впевненістю сказати що немає кращого інструменту для розробки ніж бібліотека Three.js, що має свій початок ще у 2010 році. Теоретично, цілком можливо писати на WebGL з нуля, але при цьому мати великі досягнення вкрай складно. Існує велика кількість функціоналу кожного додатку WebGL, яку можливо абстрагувати для написання кращих, більш розширених та деталізованих застосунків [8].

Перспективи, які Three.js пропонує зі свого пакету без будь-яких навичок роботи в 3D, дуже важливі, коли ми хочемо створити кілька багатовимірних проектів за короткий час. До того ж, Three.js пропонує перспективу комплексних анімацій для веб-додатків, має теорію сцени та оточення для визначення областей, де можна розмістити такі 3D об’єкти, як геометрія, світло, камери тощо. Відображення та анімацію фільтрів Three.js можна застосовувати в картографуванні, дистанційному навчанні, віртуальних турах, нерухомості, музеях та багатьох інших областях.

JAMStack це новий спосіб створення веб-сайтів і додатків, що забезпечує кращу продуктивність, більш високу безпеку, нижчу вартість масштабування та кращий досвід для розробників. Об’єктивно найбільш важливим аспектом цієї методології є відсутність щільного зв’зяку між клієнтом та веб-сервером.

Використовуючи JAMStack інженери програмного забезпечення мають можливість генерувати веб-сторінки під час їх розгортання, при цьому маючи найкращу швидкість — мінімізація часу на перший байт (TTFB — time to first byte) завантаження сторінки є неперевершеною використовуючи вбудовані файли, що

надходять з CDN. До того ж, CDN ідеально підходять для масштабування та вирішення питання подачі файлів у більшій кількості місць використання.

Якщо серверні процеси абстрагуються в мікросервісні API, то площі поверхні для зовнішніх атак таким чином зменшуються. Вільне з'єднання та розділення елементів управління також дозволяють більш цілеспрямовано розробляти та налагоджувати проекти, а розширення вибору параметрів систем управління контентом для генераторів сайтів знімає необхідність підтримувати окремий стек для контенту та маркетингу.

Flutter був обраний для написання мобільної частини додатку дипломної роботи для класифікування найбільш поширених на даний момент типів 3D файлів. Завдяки своїй високій продуктивності, додаток написаний на цьому фреймворці здатний підтримувати важкий для більшості мобільних пристроїв функціонал завантаження 3D моделей. Легкість у розробці та підтримка більшості API та засобів проектування баз даних дозволили створити якісний та легкий кросплатформний мобільний застосунок без попереднього досвіду у розробці мобільних програмних продуктів.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програмний комплекс для інтерактивної роботи з 3D об’єктами складається з двох програмних компонентів — основної веб-системи та додаткового мобільного застосунку — кожен із яких можна декомпозувати на певні функціональні блоки. Головним блоком вважається середовище для взаємодії з завантаженим об’єктом, де розташований графічний інтерфейс користувача для контролю анімацій, освітлення, управління розташуванням моделі тощо. Також, одним із основних є функціонал завантаження моделі, сценарії та структура якого описана далі у звіті.

4.1 Вимоги до програмного забезпечення

Відповідно до доданої документації SRS, основними вимогами є наступні:

— Інтерфейси користувача (User Interfaces) — система повинна дозволяти повну навігацію і вибір інтерактивних дій із використанням тільки миші, комбінацій миші та клавіатури, а також сенсорних жестів (для мобільного додатку або мобільної версії веб-сайту);

— Програмні інтерфейси (Software Interfaces) — система повинна надавати детальну інформацію щодо завантаженого об’єкту, характеристику його атрибутів та розширень;

— Комунікаційні інтерфейси (Communication Interfaces) — система відображає постійний процес завантаження моделі для надання відповідної інформації користувачу

— Вимоги продуктивності (Performance Requirements) :

- Використовуючи модемне з’єднання зі швидкістю 40 Кбіт / с, усі веб-сторінки, які генеруються системою, повинні бути повністю завантажені

не більше ніж за 5 секунд, а 3D моделі завантажені користувачем повинні бути готовими до використання не більше ніж за 15 секунд;

- Відповіді на запити повинні завантажуватися на екран не більше ніж через 5 секунд після того, як користувач відправить запит.

4.2 Опис функціональності системи

Веб-система для інтерактивної роботи з 3D об’єктами містить одного головного взаємодіючого актора — користувача системою. Детальна діаграма прецедентів (або діаграма варіантів використання UseCase) мови проектування UML для описання моделі функціональних відношень між актором та прецедентами в системі представлена на рисунку 4.1.

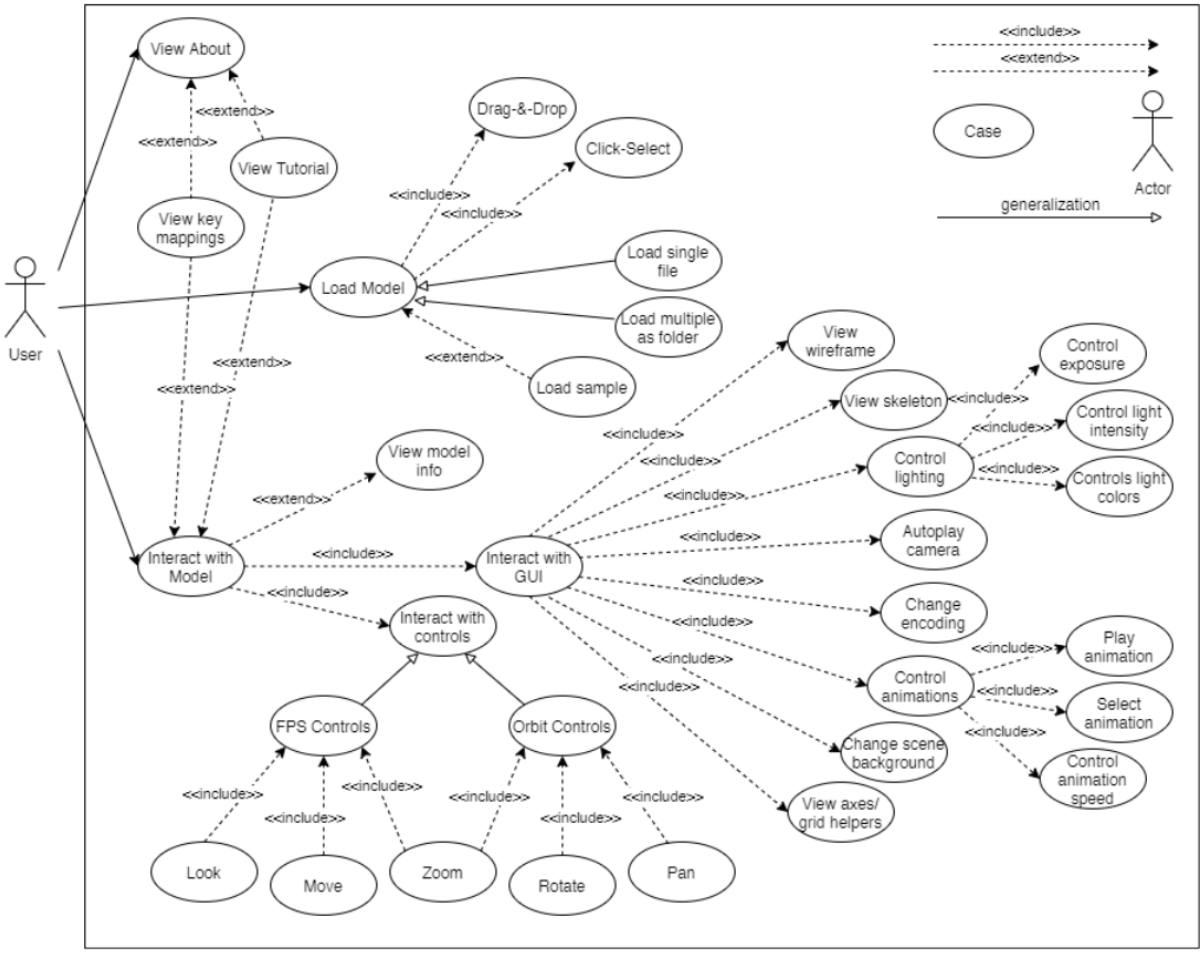

Рисунок 4.1 — Діаграма прецедентів веб-системи

4.3 Проектування дизайну системи, брендинг

Брендинг — це процес надання значенню конкретної організації, компанії, продуктів чи послуг, створюючи та формуючи бренд у свідомості споживачів. Це стратегія, розроблена організацією, щоб допомогти людям швидко визначити і відчути свій бренд, а також дати їм можливість обирати свою продукцію над конкурентами, уточнюючи конкретний бренд, що розглядується.

Крім того, що такі стратегії допомагають споживачам розрізняти подібні товари, успішні стратегії брендингу також додають репутації компанії. Цей актив може впливати на коло людей — від споживачів до службовців, інвесторів, акціонерів, постачальників та дистриб'юторів.

Елементами брендингу, які потрібно створити для покращення сприйняття клієнтами бізнесу є:

— Місія та цінності бренду, які є основою бренду;

— Вказівки щодо бренду (brand guidelines). Стратегія бренду охоплює все, що знаходиться між баченням та місією організації. Такі вказівки містяться у матеріальному документі, що відображає певні бізнес-цілі, відрізняє продукт від конкурентів, резонує з клієнтами та надає узагальнений шаблон для прийняття рішень. В даній роботі, брендовими вказівками було вирішено взяти адаптивну систему мови дизайну Material від Google, що допомагає розробникам створювати високоякісні цифрові продукти.

— Логотип — це обличчя продукту або компанії, є одним із найважливіших атрибутів бренду. Під час проектування даної системи було розглянуто багато ідей та направлень. Після першого етапу проектування та створення макетів, було обрано направлення, яке асоціюється у більшості людей з 3D зображеннями та прямими, гострими кутами, що символізують цілеспрямованість, тактичність і точність. Під час другого етапу проектування було відібрано шість претендентів на основну роль

логотипу (зображених на рисунку 4.2), які також можна використовувати у альтернативних сценаріях для просування бренду.

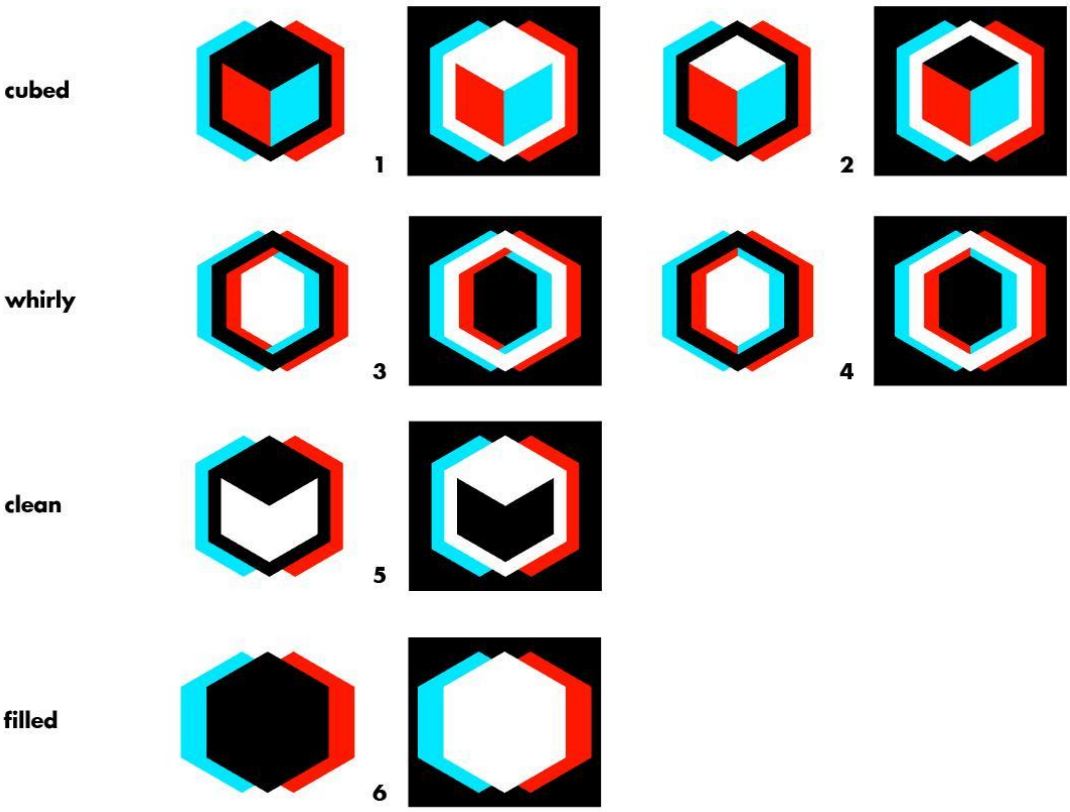

Рисунок 4.2 — Шість претендентів на роль логотипу проекту

Серед наданих варіантів було обрано один найбільш збалансовний та відповідний стратегії логотип, зображений на рисунку 4.3.

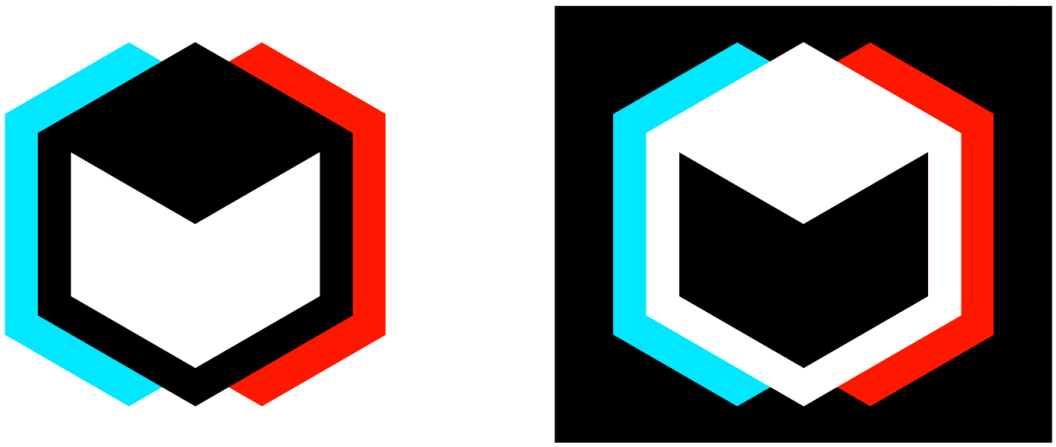

Рисунок 4.3 — Обраний логотип проекту

Палітра основних кольорів, яка була використана для проектування і також асоціюються з дефінітивом третього виміру, зображена на рисунку 4.4.

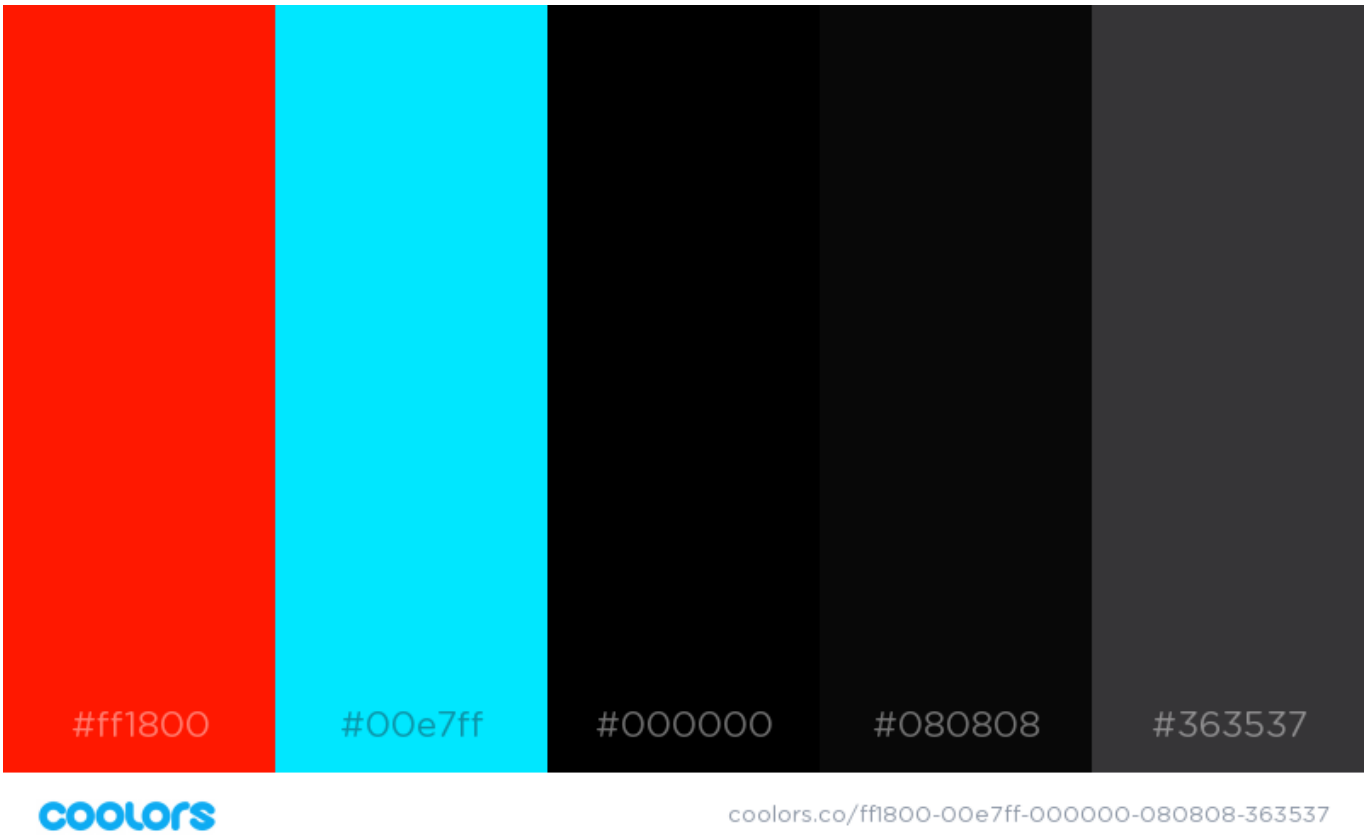

Рисунок 4.4 — Палітра основних кольорів проекту

Під час етапу проектування за допомогою програмного забезпечення Adobe Photoshop та Adobe Illustrator були створені необхідні адаптивні макети як для веб-компоненти, так і для мобільної компоненти програмного комплексу.

Макет спроектованої веб-сторінки завантаження 3D моделі зображений на рисунку 4.5.

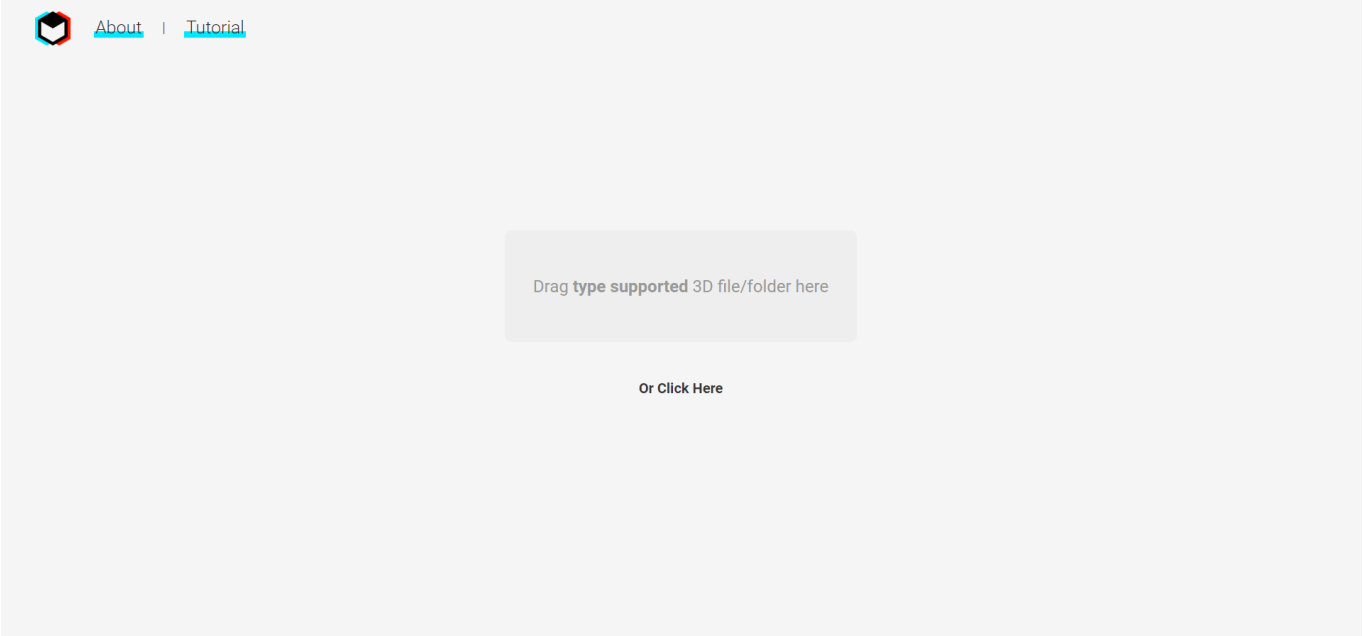

Рисунок 4.5 — Макет сторінки завантаження 3D моделі

Макет спроектованої веб-сторінки інтеракції з 3D моделлю зображений на рисунку 4.6.

Рисунок 4.6 — Макет сторінки інтеракції з 3D моделлю

Макет модального вікна ключових прив’язок (навчальний посібник) для користувача зображений на рисунку 4.7.

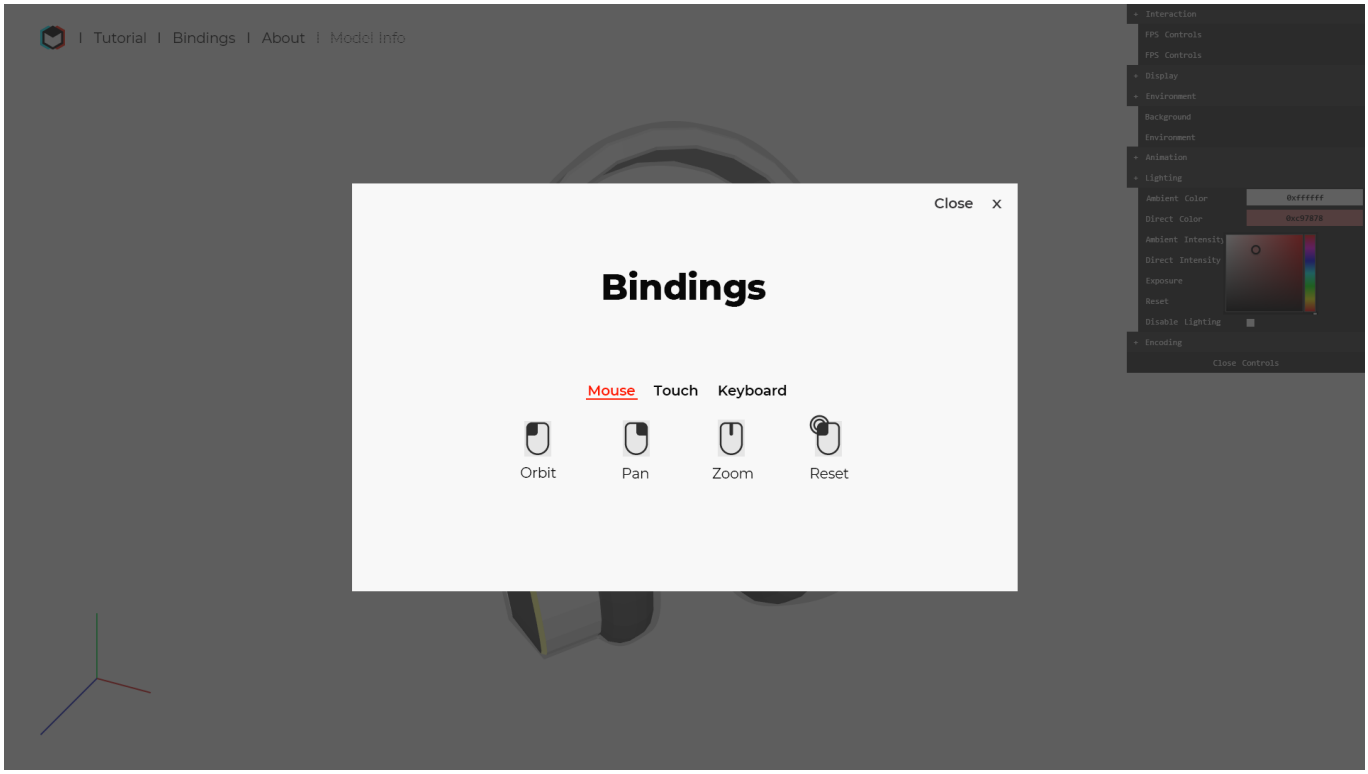

Рисунок 4.7 — Макет модального вікна ключових прив’язок

Мобільна версія застосунку також була продумана на етапі дизайну та проектування, тож були створені макети для основної сторінки мобільної версії переглядача, навігаційного меню та меню контролів інтеракції з моделлю, які зображено на рисунках 4.8, 4.9 та 4.10 відповідно.

Рисунок 4.8 — Макет мобільної версії 3D переглядача

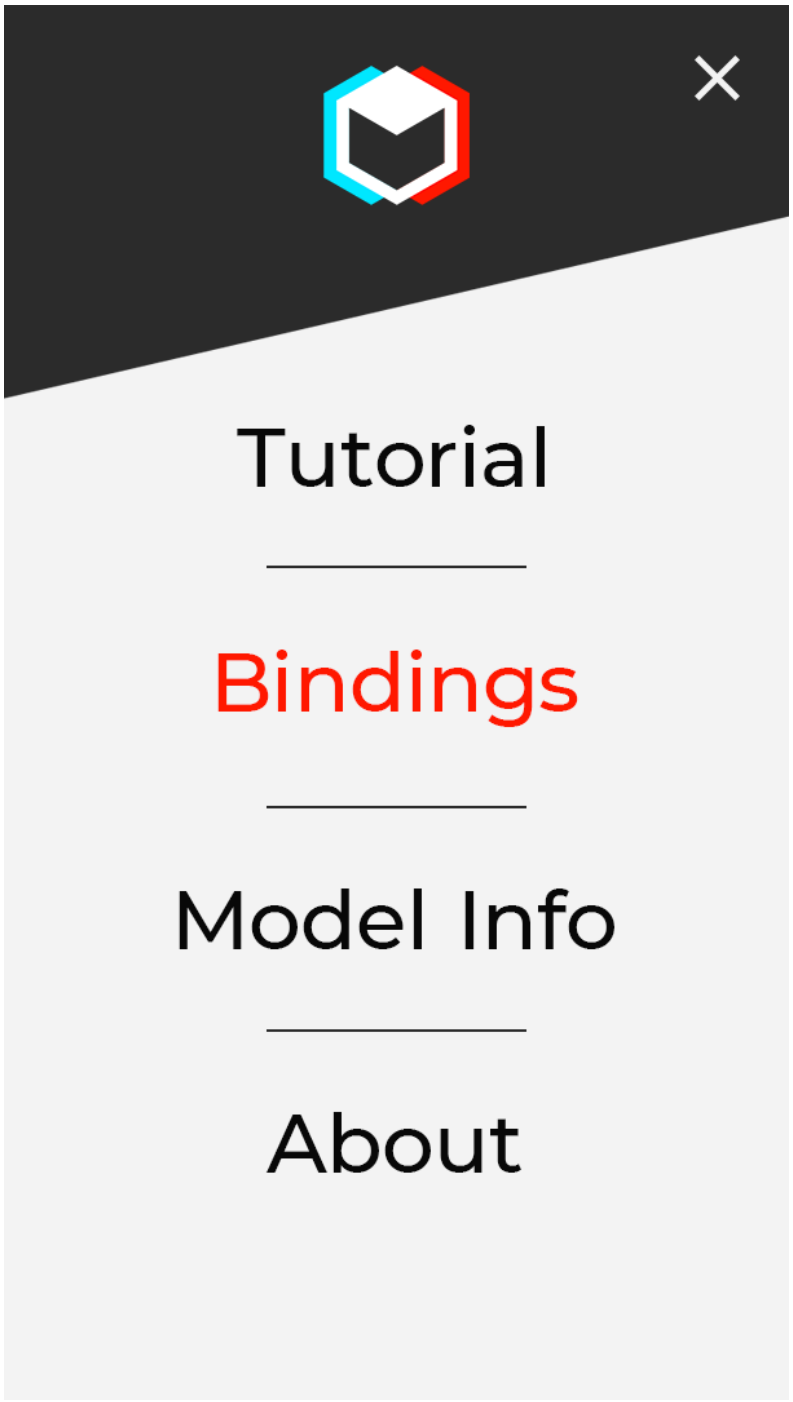

Рисунок 4.9 — Макет навігаційного меню мобільної версії переглядача

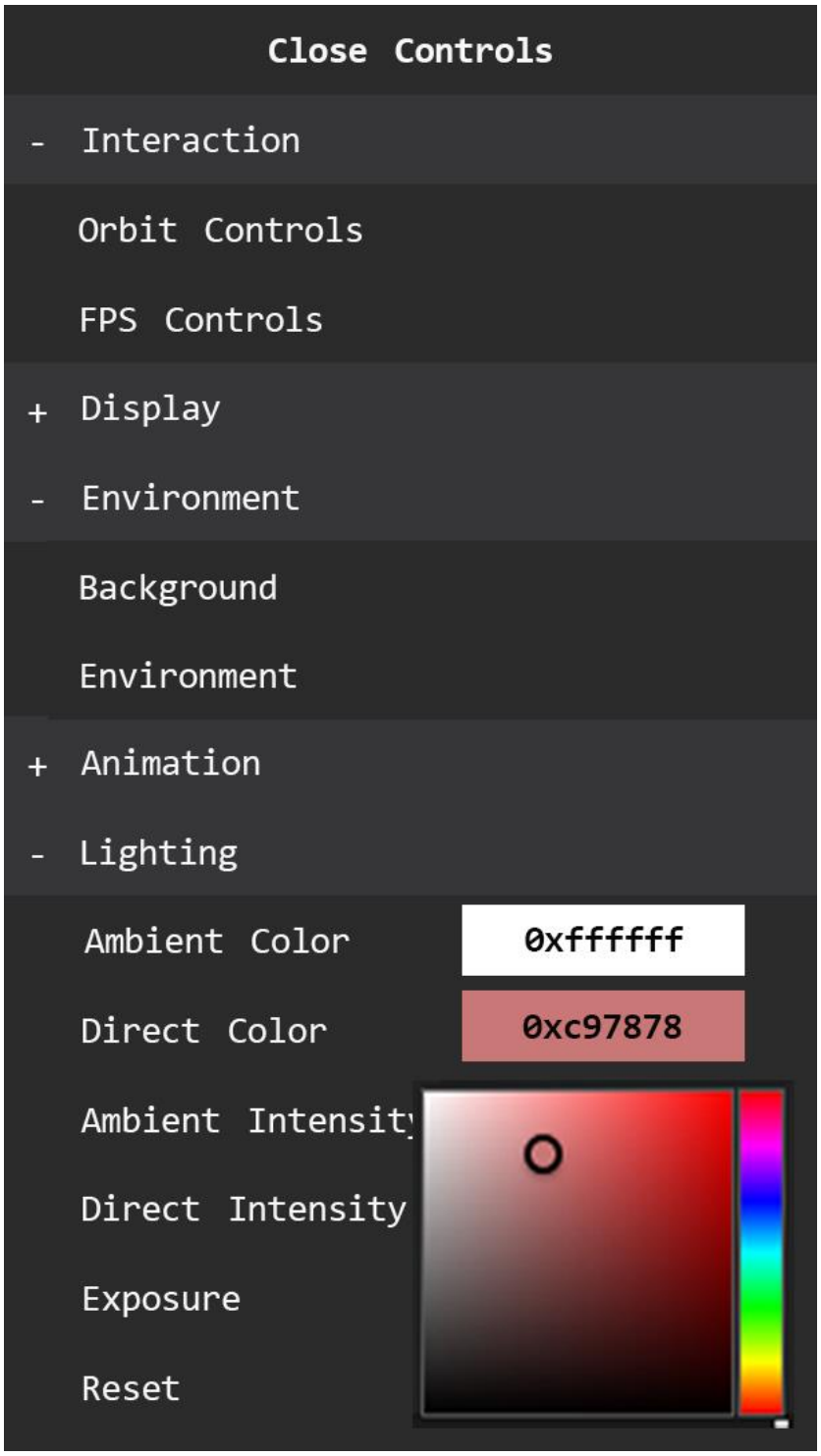

Рисунок 4.10 — Макет меню контролів мобільної версії переглядач

4.4 Архітектура веб-системи

4.4.1 Загальна структура веб-системи

Структура інтерактивної веб-системи та взаємодіючі компоненти фреймворку Vue.js та бібліотек Vuex та Three.js зображені на діаграмі класів мови проектування UML на рисунку 4.11.

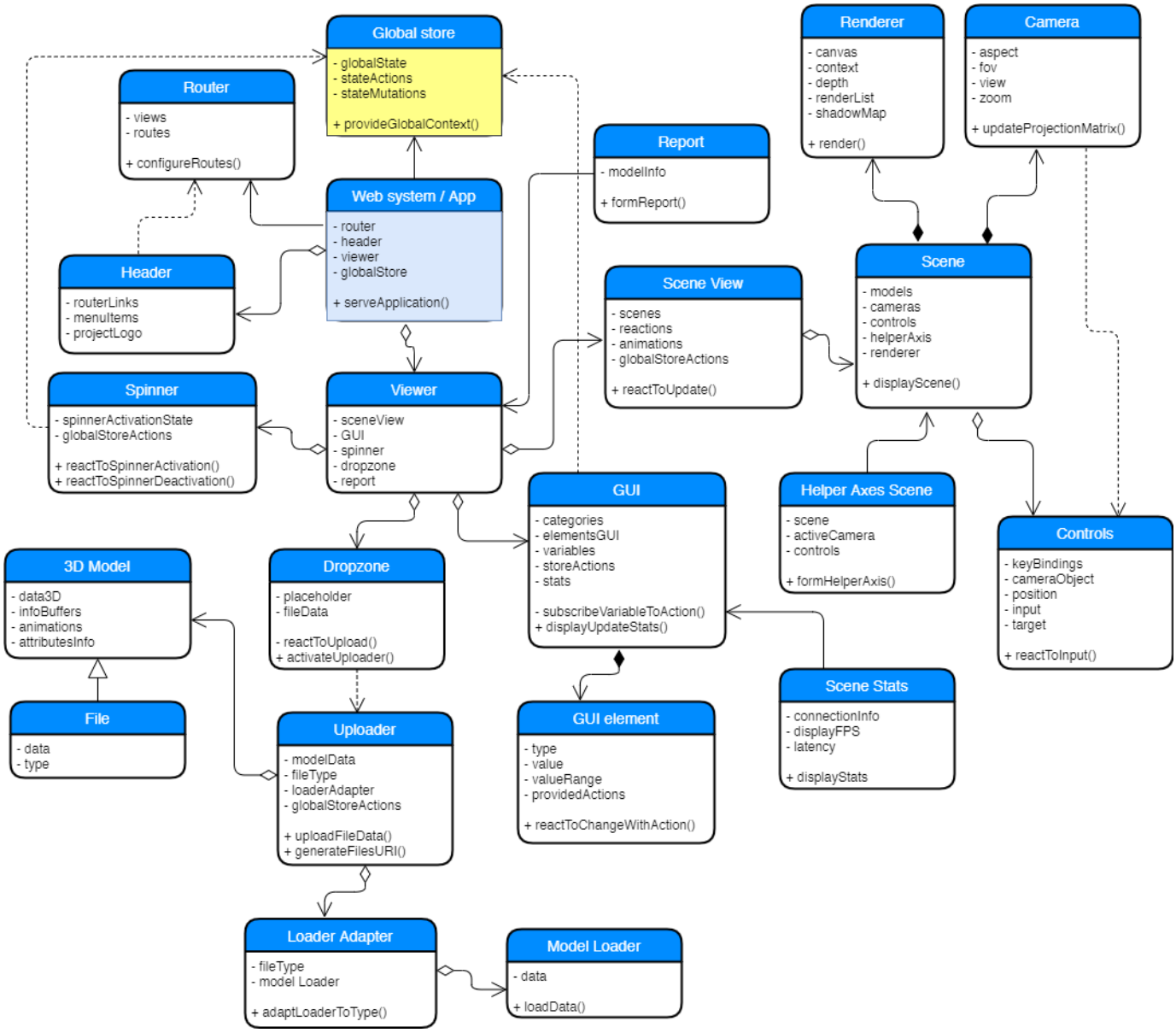

Рисунок 4.11 — Структура інтеракційної веб-системи

Під час роботи з програмою, користувач взаємодіє із клієнтським додатком (веб-системою або нативним мобільним застосунком відповідно). На рівні користувача реалізується інтерфейс, за допомогою якого налаштовується програма, переглядається інформація про проект або завантажену 3D модель та здійснюється безпосередня інтеракція із нею. Система також надає можливість проходження вбудованого навчального посібника для ознайомлення із функціональністю програми, ключовими прив’язками тощо.

Ключовими аспектами архітектури веб-системи для інтеракції з 3D об'єктами є безпосереднє завантаження моделі визначеного, підтримуваного системою розширення. Визначення підтримуваних файлів проводилося як завершальний аналізу етап поточного стану використання розширень (типів) 3D файлів. Продуктом такого аналізу також слід вважати основу додаткового мобільного застосунку, що має на меті класифікування згаданих типів та розбір і надання інформаціїї про кожного з них.

4.4.2 Завантаження 3D моделі

Веб-система дозволяє користувачам завантажувати 3D моделі як у вигляді поодиноких файлів, так і у вигляді папок із багатьма файлами та ресурсами. Завантаження може здійснюватися за рахунок подій Click-Select або Drag-and-Drop (у випадку останньої, активною площиною вважається уся площина головної сторінки додатку). Завантажити модель також можна з локального сховища, з переліку раніше завантажених моделей та з сторонньої бібліотеки 3D файлів.

Послідовність дій користувача при завантаженні певної 3D моделі зображені на діаграмі послідовностей мови проектування UML на рисунку 4.12.

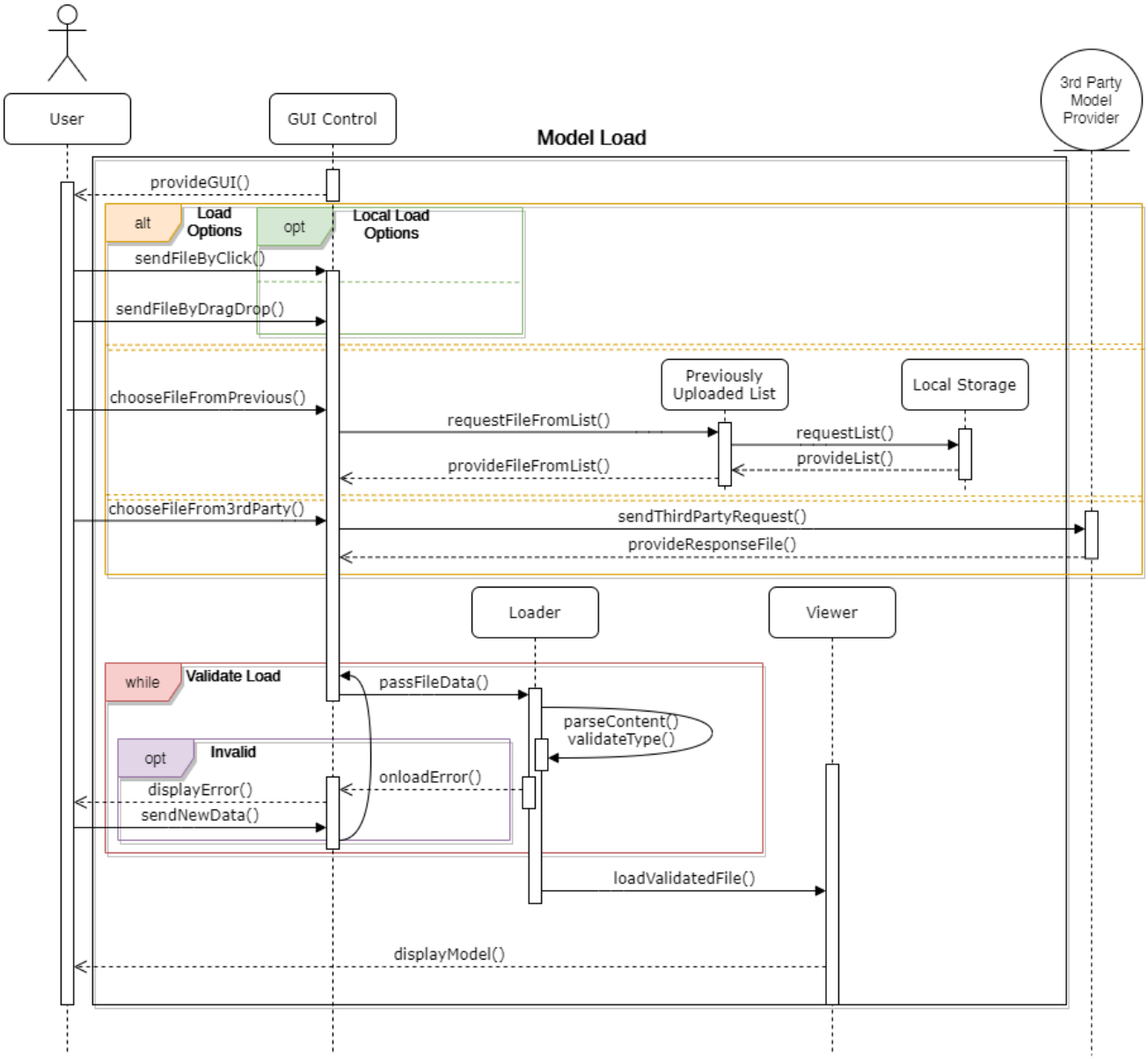

Рисунок 4.12 — Діаграма послідовностей завантаження 3D моделі

4.4.3 Інтеракція з 3D моделлю

Веб-система надає можливість різноманітної інтерактивної роботи із завантаженою моделлю. Інтеракція поділяється на 3 категорії за способом контролю:

— інтеракція із графічним інтерфейсом користувача (GUI);

— інтеракція за допомогою апаратного контролю;

— перегляд детальної інформації про модель.

4.5 Архітектура мобільного застосунку

Структура мобільного додатку та його взаємодіючі компоненти (застосунок-класифікатор найпопулярніших 3D файлів сьогодення, 3D переглядач та база даних MongoDB) зображені на діаграмі розгортання мови проектування UML на рисунку 4.13.

Під час роботи з мобільним програмним комплексом, користувач взаємодіє із інтерфейсом, написаним на фрейморку Flutter, за допомогою якого налаштовується програма, застосовується класифікатор 3D типів файлів для отримання необхідної користувачеві інформації, проглядається інформація про проект, завантажену 3D модель або навчальний посібник по роботі із застосунком та здійснюється безпосередня інтеракція із моделлю.

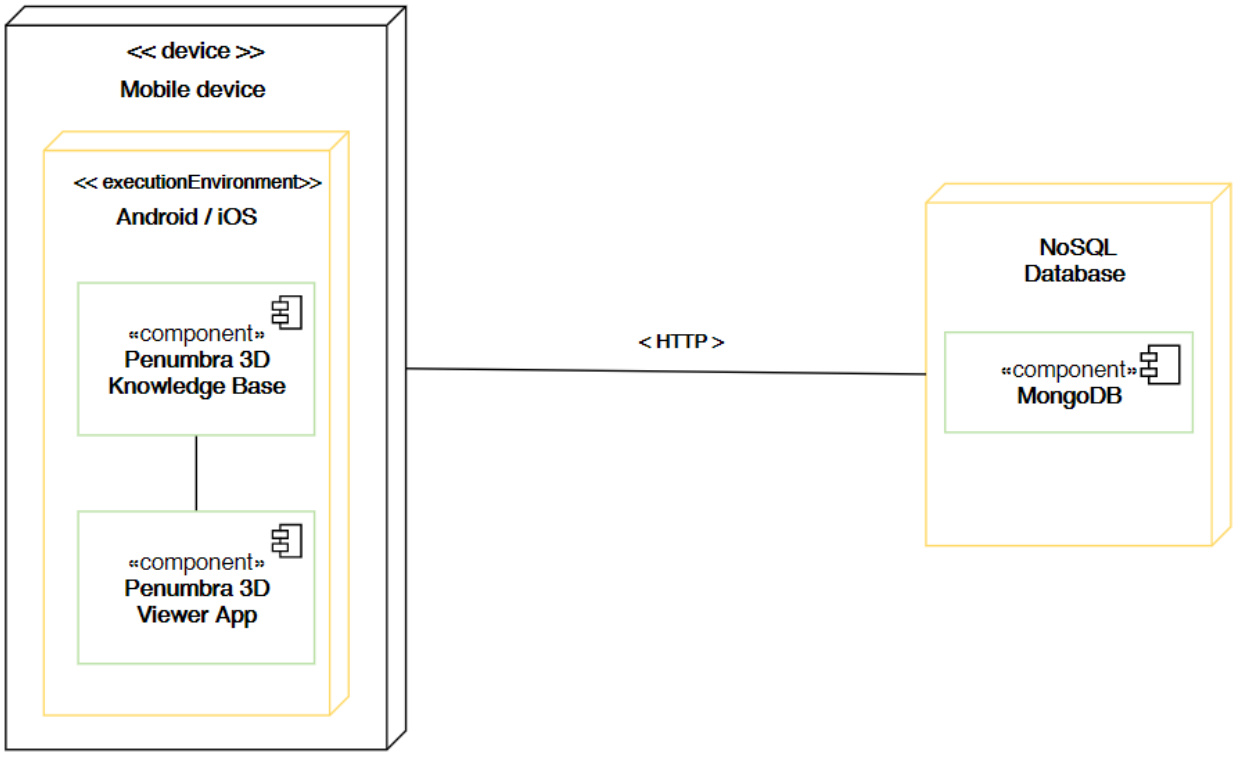

Рисунок 4.13 — Діаграма розгортання мобільного додатку

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНИМ КОМПЛЕКСОМ

5.1 Системні вимоги та інсталяція

Розроблений програмний комплекс є кросплатформеним, адже веб-система використовує новітні веб-технології, підтримувані поточними браузерами, а мобільний застосунок, в свою чергу, побудований використовуючи Flutter від Google — набір для розробки програмного забезпечення з відкритим кодом, який використовується для кросплатформеної розробки нативних додатків для Android, iOS, Windows, Mac, Linux і вебу.

5.2 Сценарій роботи користувача та функціональні вимоги

5.2.1 Завантаження 3D моделі

Послідовність стимулу / відповіді:

Стимул: Користувач за допомогою drag-and-drop пересуває файл або папку, що містить 3D модель.

Відповідь: Система завантажує модель, якщо її тип є одним із підтримуваних, надає змогу користувачи інтерактивно взаємодіяти з моделлю.

Стимул: Користувач за допомогою click-select обирає з провідника файл або папку, що містить 3D модель.

Відповідь: Система завантажує модель, якщо її тип є одним із підтримуваних, надає змогу користувачи інтерактивно взаємодіяти з моделлю.

Стимул: Користувач клікає на посилання до зразкової моделі.

Відповідь: Система завантажує архів, що містить зразкову 3D модель.

Стимул: Користувач обирає модель із переліку вже завантажених моделей.

Відповідь: Система завантажує завантажує 3D модель, якщо вона ще наявна в локальному сховищі та шлях до неї у файловій системі не змінився. У іншому випадку система відображає помилку.

Функціональними вимогами завантаження 3D моделі є:

— Система має дозволити користувачеві завантажувати моделі підтримуваних форматів;

— Система має вміти розрізняти 3D моделі за їх форматами та обирати необхідний завантажуючий модуль;

— Система має дозволяти користувачеві завантажувати наступну модель, зберігаючи при цьому попередньо встановлені конфігурації;

— Система повинна надавати усе робоче вікно програми як поле для завантаження об’єкту;

— Система має зберігати певний перелік завантажених моделей у локальному сховищі користувача.

5.2.2 Інтеракція з 3D моделлю

Послідовність стимулу / відповіді:

Стимул: Користувач використовує мишу та клавіатура (або сенсорні жести для мобільних пристроїв) для управління камерою та контролерами сцени.

Відповідь: Система здійснює ререндер сцени.

Стимул: Користувач вносить зміни в атрибути графічного інтерфейсу користувача.

Відповідь: Система реагує на зміни відповідними підписаними на зміни методами, які посилають запит на ререндер сцени.

Стимул: Користувач клікає на пункт меню “Інформація про модель”.

Відповідь: Якщо модель успішно завантажена, система надає детальну інформація про атрибути поточної моделі.

Функціональними вимогами інтеракції з 3D моделлю є:

— Система має підтвердити успішне завершення завантаження моделі;

— Система повинна надавати інформацію про моделі лише підтримуваних типів 3D файлів;

— Система має бути адаптивною для різних моделей та коригувати елементи контролю для комфортного досвіду користувача (UX — user experience);

— Система має надавати список лише тих змінних графічного інтерфейсу користувача, які здатні вплинути на модель (наприклад, система повинна сховати категорію “Анімації” для моделей розширення *.obj*).

Безпосередня взаємодія з завантаженою моделлю включає в себе наступне:

— Зміна способу контролювання оточенням (“Орбітальне” управління або управління “Від першої особи”);

— Перегляд скелету та каркасу моделі, рисунок 5.1;

— Управління світлом, рисунок 5.2:

- зміна експозиції;
- зміна інтенсивності світла;
- зміна кольорів оточуючого світла.

— Автовідтворення камери;

— Зміна методу кодування текстур, рисунок 5.3;

— Управління анімаціями моделі, рисунок 5.4:

- запуск чи зупинка анімації;
- вибір анімації, у випадку коли модель має декілька анімацій;
- управління швидкістю програвання анімації.

— Зміна 3D оточення моделі, рисунок 5.5;

— Додавання до оточення допоміжних осей координат і сіток, рисунок 5.6.

Рисунок 5.1 — Перегляд каркасу та скелету моделі

Рисунок 5.2 — Управління світлом

Рисунок 5.3 — Зміна методу кодування текстур

Рисунок 5.4 — Управління анімаціями

Рисунок 5.5 — Зміна 3D оточення моделі

Рисунок 5.6 — Допоміжні осі координат і сітки

5.2.3 Інтеракція з мобільним застосунком

Мобільний застосунок дозволяє користувачам взаємодіяти з базою знань та ресурсів щодо стану типів 3D файлів сьогодення.

На рисунку 5.7 зображена головна сторінка застосунку із переліком типів 3D файлів та короткою інформацією про кожен з них. Типи відсортовано за популярністю починаючи з найбільш популярного на даний момент типу Wavefront OBJ (розширення .obj).

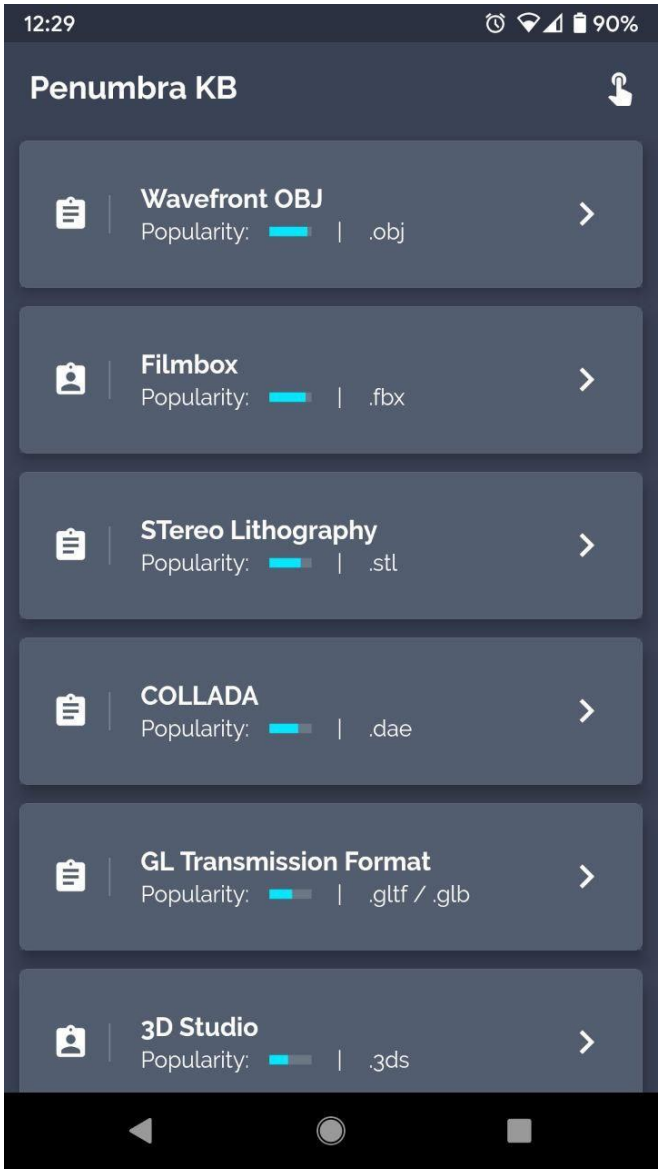

Рисунок 5.7 — Головна сторінка, перелік типів 3D файлів

На рисунку 5.8 зображено детальний опис обраного типу, його розширення, індикатор рейтингу популярності, способи та сфери використання та тип володіння (пропрієтарність типу). Надано детальну інформацію про історію та особливості

типу у короткій формі. Користувач також може отримати більше інформації та зовнішні ресурси натиснувши на кнопку “Дізнатися більше”.

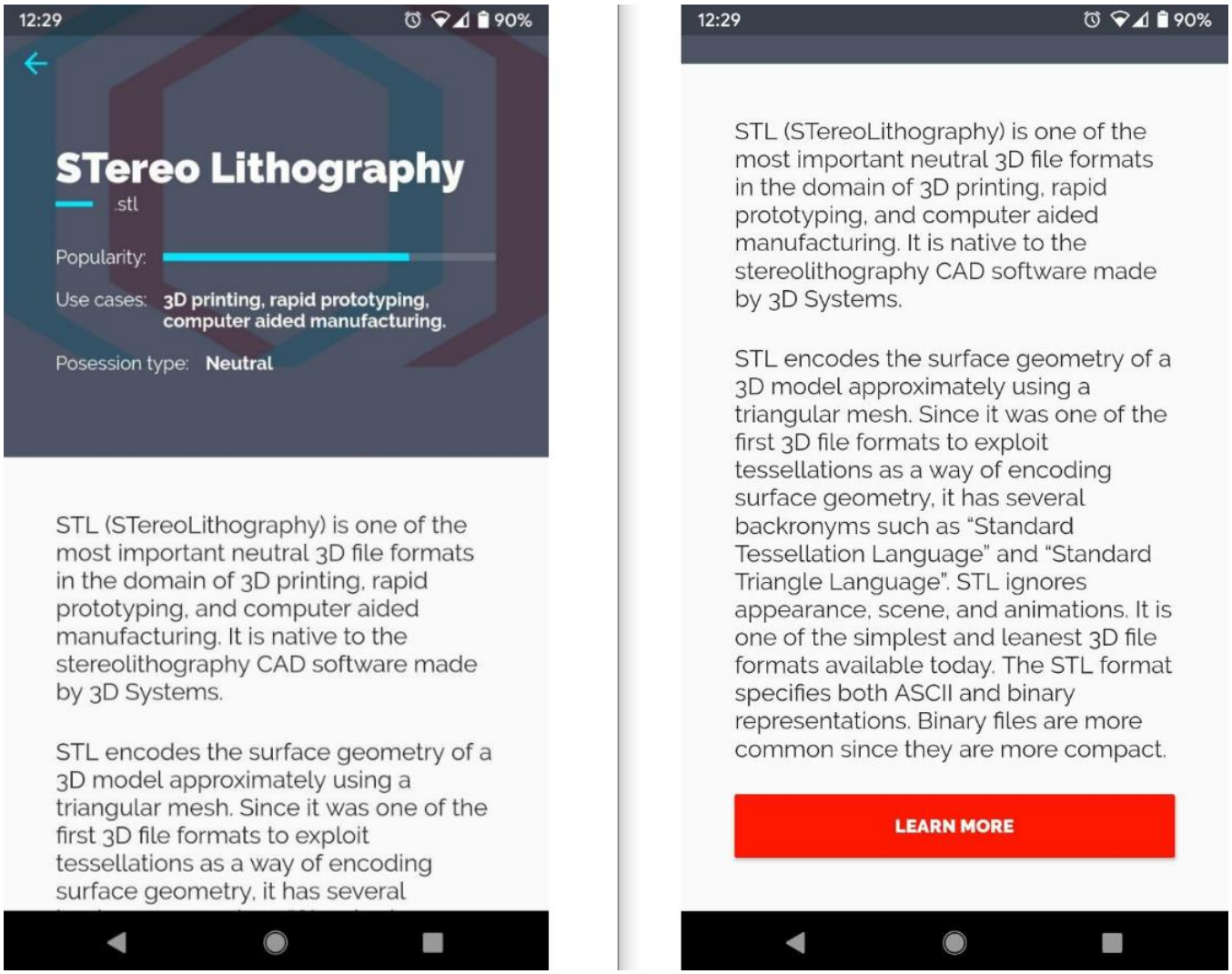

Рисунок 5.8 — Сторінка детальної інформації про тип

Користувач має можливість використати основний програмний продукт дипломної роботи — 3D переглядач для інтерактивної роботи з 3D моделями — натиснувши на навігаціонну кнопку на основній сторінці додатку.

ВИСНОВКИ

Під час аналізу поточного стану сфер 3D моделювання та веб-розробки було досліджено системи, аналогічні до розробленої. Такі реалізовані продукти було детально зіставлено і розібрано їх сильні та слабкі сторони. Таким чином, виявилися недоліки цих програмних засобів, насамперед їх громіздкість, переповненість абстрактними або неважливими специфічними функціональними можливостями, складність у налаштуванні або явну залежність від більш об’ємних, загальних та важких систем 3D моделювання [21]. Також, було виявлено, що багато з таких систем орієнтуються лише на певну частину загальної клієнтської бази та можуть бути доступними лише для певних операційних систем [22].

Описаний у даному звіті програмний комплекс призначений саме для цілі інтеракційної взаємодії з 3D об’єктами і відрізняється від конкуруючих продуктів легкістю у використанні, можливістю використання на будь-якій операційній системі, швидкою взаємодією та легким у розумінні графічним інтерфейсом користувача. Створений продукт також слідує принципам методології JAMStack, що дає відсутність щільного зв’язку між клієнтом та веб-сервером і швидкість розробки та подальшого розгортання програмного засобу.

Був проведений аналіз поточних засобів розробки веб-застосунків, обґрунтовано вибір технологій та архітектури, на якій побудована система. Була спроектована архітектура програмного комплексу та проведена брендинг-стратегія продукту, з подальшим проектуванням макетів та логотипу. Були проведені тестові сценарії використання користувачем створеного продукту, які були успішно завершені. Слід зазначити, що внаслідок аналізу поточного стану розширень 3D файлів був додатково створений кросплатформений мобільний застосунок для класифікації таких типів та для мобільної, нативної версії веб-системи.

Результати роботи демонструють що програмний комплекс можна інтегрувати в інші існуючі продукти для створення додаткового інтерфейсу взаємодії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Роджерс Д.,Адамс Дж. Математические основы машинной графики. — 2-е вид. — М. : Вільямс, 2001.

2. So Say We All: The Visual Effects of "Battlestar Galactica" [Електронний ресурс] — 7 Вересня, 2008 — https://bit.ly/2XU9h0t (дата звернення: 25.02.2020).

3. Г. Снук. 3D ландшафти в реальному часі на C + + і DirectX 9. — 2-е вид. — М. : Кудиц-прес, 2007.

4. Д. Херн, М. П. Бейкер. Комп'ютерна графіка й стандарт OpenGL. — 3-е вид. — М. : Вільямс, 2005.

5. Райт, Пол К. Производство 21-го века. — New-Jersey: Prentice-Hall Inc, 2001.

6. Слюсар, В.И.. Фаббер-технологии. Новое средство трехмерного моделирования. [Електронний ресурс] // Электроника: наука, технология, бизнес. — 2003 — http://www.electronics.ru/files/article_pdf/1/article_1269_632.pdf2003 (дата звернення: 11.04.2020).

7. The Most Common 3D File Formats [Електронний ресурс] // Dibya Chakravorty — Серпень, 2019. — https://all3dp.com/3D file-format-3D files-3D printer-3D cad-vrml-stl-obj (дата звернення: 25.04.2020).

8. Дж. Лі, Б. Уер. Тривимірна графіка та анімація. — 2-е вид. — М. : Вільямс, 2002.

9. 3D virtual reality models help yield better surgical outcomes: Innovative technology improves visualization of patient anatomy, study finds. [Електронний ресурс] // ScienceDaily — 18 Вересня, 2019 — https://bit.ly/3004QnD (дата звернення: 06.03.2020).

10. Е. Енджел. Інтерактивна комп'ютерна графіка. Вступний курс на базі OpenGL. — 2-е вид. — М. : Вільямс.

11. Jason van Gumster. Blender For Dummies. — 3-е вид. — 27 квітня 2015.

12. Chua, C. K; Leong, K. F.; Lim, C. S. Rapid Prototyping: Principles and Applicat3ons — 2-е вид. — World Scientific Publishing Co, 2003.

13. STL 2.0: A Proposal for a Universal Multi-Material Additive Manufacturing File Format // Hiller, Jonathan D.; Lipson, Hod.. Cornell University — 2009 — (PDF) https://sffsymposium.engr.utexas.edu/Manuscripts/2009/2009-23-Hiller.pdf (дата звернення: 10.04.2020).

14. What is 3D Printing? The definitive guide. [Електронний ресурс] // 3D Hubs. — 2017 — https://www.3dhubs.com/what-is-3D printing (дата звернення: 16.02.2020).6. Khronos Group glTF Briefing [Електронний ресурс] // Khronos Group — Вересень, 2016. — https://bit.ly/2Wr5PuW (дата звернення: 03.05.2020).

15. Khronos Group glTF Briefing [Електронний ресурс] // Khronos Group — Вересень, 2016. — https://bit.ly/2Wr5PuW (дата звернення: 03.05.2020).

16. An Overview of 3D Data Content, File Formats and Viewers [Електронний ресурс] // Kenton McHenry and Peter Bajcsy — 31 жовтня 2008 — https://www.archives.gov/files/applied-research/ncsa/8-an-overview-of-3D data-content-file-formats-and-viewers.pdf (дата звернення: 02.05.2020).

17. The JAMStack in 2020: Why (And How) to Get Started [Електронний ресурс] // Mathieu Dionne — Січень, 2020 — https://snipcart.com/blog/jamstack (дата звернення: 29.04.2020).

18. Mwalongo, Finian; Krone, Michael; et al. Visualization of molecular structures using state-of-the-art techniques in WebGL. — International Conference on 3D Web Technology. — Ванкувер, Серпень 2014.

19. В. П. Іванов, А. С. Батраков. Тривимірна комп'ютерна графіка / Під ред. Г. М. Поліщука. — М. : Радіо та зв'язок, 1995.

20. Інженерна та комп’ютерна графіка : підруч. для студ. ВНЗ / В. Є. Михайленко, В. В. Ванін, С. М. Ковальов. — 5-те вид. — К. : Каравела, 2010.

21. Three.js Fundamentals [Електронний ресурс] // ThreeJSFundamentals Contributors — Січень, 2020 — https://threejsfundamentals.org/threejs/lessons/threejs-fundamentals.html (дата звернення: 06.05.2020).

22. Best Cross-Platform Mobile Development Tools in 2020 [Електронний ресурс] // Anastasia Galadzhii — 10 Жовтня, 2019 — https://litslink.com/blog/best-cross-platform-mobile-development-tools-in-2019 (дата звернення: 16.05.2020).

ДОДАТОК А

Веб-система для інтерактивної роботи з 3D об’єктами

Специфікація

УКР.НТУУ "КПІ". ТІ6294_20Б

Аркушів 2

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6294_20Б	ТІ62_Стеценко_Записка.doc	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6294_20Б	Viewer.vue	Основний компонент
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6294_20Б	Scene.vue	Компонент 3D сцени переглядача
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6294_20Б	store/index.js	Компонент Vuex зберігання
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6294_20Б	Lightning.js	Компонент управління освітленням

ДОДАТОК Б

Веб-система для інтерактивної роботи з 3D об’єктами

Текст програми

УКР.НТУУ "КПІ". ТІ6294_20Б

Аркушів 19

Київ 2020

Viewer.vue

```
<template>
  <main id="viewer" class="viewer">
    <Modal v-if="showTutorial" @close="toggleModalTutorial">
      <Tutorial slot="body" />
    </Modal>
    <Modal v-if="showModelInfo" @close="toggleModalModelInfo">
      <ModelInfo slot="body" />
    </Modal>

    <div class="dropzone" id="dropzone" :ref="'dropzone'">
      <GUI :class="{ 'no-display': !isLoaded }" />
      <Scene :class="{ 'no-display': !isLoaded }" />
      <UploadPlaceholder :class="{ 'no-display': isLoaded }" />
    </div>
    <Spinner v-if="showSpinner" :ref="'spinner'" />
  </main>
</template>

<script>
import { mapState, mapGetters, mapActions } from 'vuex'

import { SimpleDropzone } from 'simple-dropzone'

import GUI from '@/components/GUI.vue'
import Scene from '@/components/Scene/Scene.vue'
import UploadPlaceholder from '@/components/UploadPlaceholder.vue'
import Spinner from '@/components/Spinner.vue'
import Modal from '@/components/modals/Modal.vue'
import ModelInfo from '@/components/modals/ModelInfo.vue'
import Tutorial from '@/components/modals/Tutorial.vue'

import { ALLOW_FILE_TYPE } from '@/utils/supportedTypes.js'

export default {
  name: 'viewer',

  components: {
    GUI,
    Scene,
    UploadPlaceholder,
    Spinner,
    Modal,
    ModelInfo,
    Tutorial
  },

  computed: {
    ...mapState(['showSpinner', 'showModelInfo', 'showTutorial'])
```

```
},

data() {
  return {
    isLoaded: false
  }
},

mounted() {
  const dropzoneController = new SimpleDropzone(
    // Unable to use refs due to file-input and drop-zone being
    // on completely different component levels
    document.getElementById('dropzone'),
    document.getElementById('file-input')
  )

  dropzoneController
    .on('drop', ({ files }) => {
      console.log('CHECKING DROPZONE FILESOBJ...')
      console.log(files)
      this.load(files)
    })
    .on('dropstart', () => {
      this.$store.commit('ACTIVATE_SPINNER')
    })
    .on('droperror', () => {
      this.$store.commit('DEACTIVATE_SPINNER')
    })
},

methods: {
  toggleModalTutorial() {
    this.$store.commit('TOGGLE_MODAL_TUTORIAL')
  },
  toggleModalModelInfo() {
    this.$store.commit('TOGGLE_MODAL_MODEL_INFO')
  },

  load(fileMap) {
    let rootFile
    let rootPath
    let fileType

    console.log('Map of file(s)...')
    console.dir(fileMap)

    Array.from(fileMap).forEach(([path, file]) => {
      if (file.name.toLowerCase().match(ALLOW_FILE_TYPE)) {
        rootFile = file
```

```
        // Get file type to determine model loader type later
        fileType = file.name
          .split('.')
          .pop()
          .toLowerCase()

        rootPath = path.replace(file.name, '')
      }
    })

    if (!rootFile) {
      // TODO: create a more legible error string
      this.onError(`No ${ALLOW_FILE_TYPE} asset found.`)
    }

    this.view(rootFile, rootPath, fileMap, fileType)
  },

  view(rootFile, rootPath, fileMap, fileType) {
    const fileURL =
      typeof rootFile === 'string' ? rootFile : URL.createObjectURL(rootFile)

    // Save fileData to global VueX storage
    const fileData = { fileURL, rootPath, fileMap, fileType }
    this.$store.dispatch('saveFileData', fileData)

    // Save fileData as model to global VueX storage
    const modelData = { fileName: rootFile.name, fileType }
    this.$store.dispatch('saveModelToHistory', modelData)

    this.isLoaded = true

    console.log('LocalStorage Model History...')
    console.dir(JSON.parse(localStorage.modelHistory))
  },

  onError(error) {
    let message = (error || {}).message || error.toString()
    if (message.match(/ProgressEvent/)) {
      message =
        'Unable to retrieve this file. Check JS console and browser network tab.'
    } else if (message.match(/Unexpected token/)) {
      message = `Unable to parse file content. Verify that this file is valid. Error: "${message}"`
    } else if (error && error.target && error.target instanceof Image) {
      message = 'Missing texture: ' + error.target.src.split('/').pop()
    }
    window.alert(message)
    console.error(error)
  }
}
```

```
}
</script>

<style lang="scss">
.viewer {
  display: flex;
  flex-direction: column;
  width: 100vw;
  flex-grow: 1;
  position: relative;
}

.dropzone {
  display: flex;
  justify-content: center;
  align-items: center;
  width: 100%;
  height: 100%;
}
</style>
```

Scene.vue

```
<template>
  <div class="scene-wrapper">
    <div id="scene" class="scene" :ref="'scene'"></div>
    <div id="axes" class="axes" :ref="'axes'"></div>
  </div>
</template>

<script>
import { mapState, mapGetters, mapActions } from 'vuex'

import { traversePrint, traverseMaterials } from '@/utils/traverse.js'
import { snakeCapsToCamel } from '@/utils/snakeCapsToCamel.js'
import * as loaders from '../../loaders'

import updateDisplay, { addAxesScene } from './Display'
import updateControls from './Controls'
import updateEnvironment from './Environment'
import updateEncoding from './Encoding'
import updateLighting from './Lighting'
import updateAnimation, {
  setClips,
  playClips,
  playAnimations
} from './Animation'

import * as THREE from 'three'
import { OrbitControls } from 'three/examples/jsm/controls/OrbitControls'
```

```
import { FirstPersonControls } from 'three/examples/jsm/controls/FirstPersonControls'
import { PointerLockControls } from 'three/examples/jsm/controls/PointerLockControls'
import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader.js'
import { FBXLoader } from 'three/examples/jsm/loaders/FBXLoader'
import { DRACOLoader } from 'three/examples/jsm/loaders/DRACOLoader.js'

export default {
  name: 'scene',

  computed: {
    ...mapState([
      'fileURL',
      'rootPath',
      'fileMap',
      'fileType',
      'foldersGUI',
      'sceneState'
    ])
  },

  data() {
    return {
      defaultCamera: null,
      scene: null,
      renderer: null,
      controls: null,

      content: null,

      // Texture roughness values
      pmremGenerator: null,

      // Lighting
      lights: [],

      // Animation
      clips: null,
      mixer: null,
      animControls: [],
      clock: null,

      // Skeleton
      skeletonHelpers: [],

      // Grid
      gridHelper: null,

      // Axes
      axesHelper: null,
      axesRenderer: null,
```

```
    axesCamera: null,
    axesCorner: null,
    axesScene: null
  }
},

methods: {
  reset() {
    this.$refs['scene'].innerHTML = ''
    this.$refs['axes'].innerHTML = ''
    this.clock = new THREE.Clock()
    this.lights = []
    this.gridHelper = null
    this.axesHelper = null
    this.content = null // TODO: needed??
  },

  init() {
    const el = this.$refs['scene']

    // Scene
    this.scene = new THREE.Scene()

    // Camera
    // const fov = options.preset === Preset.ASSET_GENERATOR ? (0.8 * 180) / Math.PI : 60
    this.defaultCamera = new THREE.PerspectiveCamera(
      60, // fov
      el.clientWidth / el.clientHeight,
      0.01,
      1000
    )
    this.scene.add(this.defaultCamera)

    // Renderer
    this.renderer = new THREE.WebGLRenderer({ antialias: true })
    this.renderer.physicallyCorrectLights = true
    this.renderer.outputEncoding = this.sceneState.outputEncoding
    this.renderer.setClearColor(0xcccccc)
    this.renderer.setPixelRatio(window.devicePixelRatio)
    this.renderer.setSize(el.clientWidth, el.clientHeight)
    this.renderer.shadowMap.enabled = true

    this.pmremGenerator = new THREE.PMREMGenerator(this.renderer)
    this.pmremGenerator.compileEquirectangularShader()

    // Controls
    this.updateControls()
    this.controls.screenSpacePanning = true // camera pans in screen space
    this.controls.autoRotate = this.sceneState.cameraAutoplay
    this.controls.autoRotateSpeed = -3
```

```
  el.appendChild(this.renderer.domElement)
  this.addAxesScene()

  // Resize resolution workaround
  // this.onWindowResize.bind(null, el),
  window.addEventListener('resize', this.onWindowResize, false)
},

animate() {
  requestAnimationFrame(this.animate)

  const delta = this.clock.getDelta()
  if (this.mixer) this.mixer.update(delta)

  this.render(delta)
},

render(delta) {
  this.renderer.render(this.scene, this.defaultCamera)

  // required if controls.enableDamping or controls.autoRotate are set to true
  if (this.controls.enabled) this.controls.update(delta)

  // Adds axisScene
  if (this.sceneState.axes) {
    this.axesCamera.position.copy(this.defaultCamera.position)
    this.axesCamera.lookAt(this.axesScene.position)
    this.axesRenderer.render(this.axesScene, this.axesCamera)
  }
},

onWindowResize() {
  // Main scene resize
  // const { clientWidth, clientHeight } = this.$refs['scene']
  const mainEl = this.$refs['scene']
  this.defaultCamera.aspect = mainEl.clientWidth / mainEl.clientHeight
  this.defaultCamera.updateProjectionMatrix()
  this.renderer.setSize(mainEl.clientWidth, mainEl.clientHeight)

  // // Axes scene resize
  // const axesEl = this.$refs['axes']
  // this.axesCamera.aspect = axesEl.clientWidth / axesEl.clientHeight
  // this.axesCamera.updateProjectionMatrix()
  // this.axesRenderer.setSize(axesEl.clientWidth, axesEl.clientHeight)

  // Controls
  if (this.sceneState.fpsControls) {
    this.controls.handleResize() // TODO: Check if the name is valid
  }
```

```
},

/**
 * object = Scene object
 */
setContent(object, clips) {
  const box = new THREE.Box3().setFromObject(object)
  const size = box.getSize(new THREE.Vector3()).length()
  const center = box.getCenter(new THREE.Vector3())

  // this.controls.reset()

  object.position.x += object.position.x - center.x
  object.position.y += object.position.y - center.y
  object.position.z += object.position.z - center.z

  // this.controls.maxDistance = size * 10

  this.defaultCamera.near = size / 1000 // 100
  this.defaultCamera.far = size * 1000 // 100
  this.defaultCamera.updateProjectionMatrix()

  // if (this.options.cameraPosition) {
  // this.defaultCamera.position.fromArray(this.options.cameraPosition)
  // this.defaultCamera.lookAt(new THREE.Vector3())
  // } else {
  this.defaultCamera.position.copy(center)
  this.defaultCamera.position.x += size / 2.0
  this.defaultCamera.position.y += size / 5.0
  this.defaultCamera.position.z += size / 2.0
  this.defaultCamera.lookAt(center)
  // }

  // this.setCamera()

  // AxesHelper
  this.axesCamera.position.copy(this.defaultCamera.position)
  this.axesCamera.lookAt(this.axesScene.position)
  this.axesCamera.near = size / 100
  this.axesCamera.far = size * 100
  this.axesCamera.updateProjectionMatrix()
  this.axesCorner.scale.set(size, size, size)

  // this.controls.saveState()
  // object.scale = new THREE.Vector3(100, 100, 100)

  this.scene.add(object)
  this.content = object

  this.$store.commit('SET', { disableLighting: false })
```

```
  this.content.traverse(node => {
    if (node.isMesh) {
      node.material.depthWrite = !node.material.transparent
    } else if (node.isLight) {
      this.$store.commit('SET', { disableLighting: true })
    }
  })

  //

  this.updateLighting()
  this.updateEncoding()
  this.updateDisplay()
  this.updateAnimation()
  this.updateEnvironment()

  this.setClips(clips)
  this.resetGUI()
  window.content = this.content
  console.info('THREE.Scene exported as `window.content`')
  // console.dir(this.content)
  traversePrint(this.content)
},

setClips(clips) {
  setClips(this.$data, clips)
},

playClips() {
  playClips(this.$data)
},

resetGUI() {
  this.animControls.forEach(ctrl => ctrl.remove())
  this.animControls.length = 0

  // -- Don't display the animation folder
  // ...

  // Animations playout
  if (this.clips.length) {
    // -- Display the animation folder
    // Get the folder classlist
    // Remove no-display from it
    // Commit changes to store
    // ...

    // Play the animations
    playAnimations(this.$data)
```

```
    // Use animation folder to play other clips when enabled
    // playOtherClips()
    // (Also, dont forget to add animCtrls to this.$data)
  }
},

updateEnvironment() {
  updateEnvironment(this.$data, this.sceneState, this.$store)
},

updateAnimation() {
  updateAnimation(this.$data, this.sceneState)
},

resetGUIFolders() {
  this.$store.commit('SET_FOLDER_GUI_DISPLAY', { anim: 'none' })
},

loadModel() {
  let loaderPromise

  switch (this.fileType) {
    case 'gltf':
      loaderPromise = loaders.loadGLTF(this)
      break
    case 'glb':
      loaderPromise = loaders.loadGLTF(this)
      break
    case 'fbx':
      loaderPromise = loaders.loadFBX(this)
      break
    case 'obj':
      loaderPromise = loaders.loadOBJ(this)
      break
    case 'dae':
      loaderPromise = loaders.loadDAE(this)
      break
    case 'stl':
      loaderPromise = loaders.loadSTL(this)
      break

    default:
      console.error('Unable to load the model')
      break
  }

  return loaderPromise
},
```

```
addAxesScene() {
  addAxesScene(this.$data, this.$refs['axes'])
},

updateDisplay() {
  updateDisplay(this.$data, this.sceneState)
},

updateCamera() {
  this.controls.autoRotate = this.sceneState.cameraAutoplay
},

updateControls() {
  updateControls(this.$data, this.sceneState)
},

updateEncoding() {
  updateEncoding(this.$data, this.sceneState)
},

resetLighting() {
  console.log('Calculating model info...')
  this.getModelInfo(this.scene)

  this.$store.commit('SET_DEFAULT_LIGHTING')
  this.updateLighting()
},

getModelInfo(scene) {
  const { children } = scene

  let objects = 0
  let vertices = 0
  let triangles = 0

  children.forEach(child => {
    child.traverseVisible(object => {
      objects++

      if (object.isMesh) {
        const geometry = object.geometry

        if (geometry.isGeometry) {
          vertices += geometry.vertices.length
          triangles += geometry.faces.length
        } else if (geometry.isBufferGeometry) {
          vertices += geometry.attributes.position.count

          if (geometry.index !== null) {
            triangles += geometry.index.count / 3
```

```
          } else {
            triangles += geometry.attributes.position.count / 3
          }
        }
      }
    })
  })

  const modelInfo = {
    key: 'Model Info',
    value: [
      { key: 'vertices', value: vertices },
      { key: 'triangles', value: triangles },
      { key: 'objects', value: objects }
    ]
  }
  this.$store.dispatch('saveModelInfo', modelInfo)
 },

 updateLighting() {
  updateLighting(this.$data, this.sceneState)
 }
},

watch: {
 fileURL: function() {
  // Initiate the scene
  this.reset()
  this.init()
  this.animate()

  // Same as viewer.load().catch().then()
  this.loadModel()
   // On Error handler (should be same as in Viewer's onError)
   .catch(error => {
    console.log('in error')
    let message = (error || {}).message || error.toString()
    window.alert(message)
    console.error(error)
   })
   // Then Handler
   .then(object => {
    this.getModelInfo(this.scene)

    // Cleanup
    console.log('in cleanup')
    this.$store.commit('DEACTIVATE_SPINNER')
    if (typeof this.rootFile === 'object')
     URL.revokeObjectURL(this.fileURL)
   })
```

```
    }
  },

  created() {
    this.$store.subscribe((mutation, state) => {
      // console.log(`Reacting to ${mutation.type} mutation`)

      // Assign only certain types of mutations, ALLOW pitfall
      switch (mutation.type) {
        case 'UPDATE_DISPLAY':
        case 'UPDATE_ENVIRONMENT':
        case 'UPDATE_CAMERA':
        case 'UPDATE_CONTROLS':
        case 'UPDATE_LIGHTING':
        case 'RESET_LIGHTING':
        case 'UPDATE_ENCODING':
        case 'UPDATE_ANIMATION':
        case 'PLAY_CLIPS':
          // console.log(`Reacting to ${mutation.type} mutation`)

          this[snakeCapsToCamel(mutation.type)]()
          break

        default:
          break
      }
    })
  }
}
</script>

<style lang="scss" scoped>
.scene-wrapper {
  position: relative;
  width: 100%;
  height: 100%;
}

.scene {
  display: flex;
  flex-direction: column;
  position: relative;
  width: 100%;
  height: 100%;
}

.axes {
  width: 100px;
  height: 100px;
  margin: 20px;
```

```
  padding: 0px;
  position: absolute;
  left: 0px;
  bottom: 0px;
  z-index: 10;
  pointer-events: none;
}
</style>
```

store/index.js

```
import Vue from 'vue'
import Vuex from 'vuex'

import { sRGBEncoding, LinearEncoding } from 'three'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    showSpinner: false,
    showModelInfo: false,
    showTutorial: false,

    // File loading
    fileURL: '',
    rootPath: '',
    fileMap: null,
    fileType: '',

    // File history
    // modelHistory: [],
    modelHistory: localStorage.modelHistory
      ? JSON.parse(localStorage.modelHistory)
      : [],

    // Loaded model information
    modelInfo: {},

    // Visibility control for specific GUI folders
    foldersGUI: {
      // anim: 'no-display'
      anim: null
    },

    // GUI variables
    sceneState: {
      // Interaction
      fpsControls: false,
      cameraAutoplay: false,
```

```
    // Display
    grid: false,
    axes: false,
    wireframe: false,
    skeleton: false,

    // Environment
    background: false,
    environment: 'None',

    // Animation
    playbackSpeed: 1.0,
    // actionStates: {},

    // Lighting
    ambientColor: 0xffffff,
    directColor: 0xffffff,
    ambientIntensity: 0.4,
    directIntensity: 0.8 * Math.PI,
    exposure: 1.0,
    disableLighting: false,

    // Encoding
    outputEncoding: sRGBEncoding,
    textureEncoding: sRGBEncoding
  }
},

mutations: {
  SET: (state, { key, value }) => (state[key] = value),

  SET_SCENE_PROP: (state, { key, value }) => (state.sceneState[key] = value),

  TOGGLE_MODAL_MODEL_INFO: state => {
    state.showModelInfo = !state.showModelInfo
  },
  TOGGLE_MODAL_TUTORIAL: state => {
    state.showTutorial = !state.showTutorial
  },

  ACTIVATE_SPINNER: state => {
    state.showSpinner = true
  },
  DEACTIVATE_SPINNER: state => {
    state.showSpinner = false
  },

  SAVE_FILE_DATA: (state, { fileURL, rootPath, fileMap, fileType }) => {
    state.fileURL = fileURL
```

```
    state.rootPath = rootPath
    state.fileMap = fileMap
    state.fileType = fileType
  },
  RESET_FILE_DATA: state => {
    state.fileURL = ''
    state.rootPath = ''
    state.fileMap = null
    state.fileType = ''
  },
  SAVE_MODEL_TO_HISTORY: (state, model) => {
    let history = state.modelHistory
    history.push(model)
    if (history.length > 20) {
      // trim localStorage model history up to 20 last
      history.shift()
      // history.splice(history.length - 20, 20)
    }
    localStorage.modelHistory = JSON.stringify(history)
  },

  SAVE_MODEL_INFO: (state, modelInfo) => {
    state.modelInfo = modelInfo
  },

  // GUI manipulations
  SET_DEFAULT_LIGHTING: state => {
    state.sceneState.ambientColor = 0xffffff
    state.sceneState.directColor = 0xffffff
    state.sceneState.ambientIntensity = 0.4
    state.sceneState.directIntensity = 0.8 * Math.PI
    state.sceneState.exposure = 1.0
  },

  UPDATE_CAMERA: () => console.log('~~ updateCamera notifier'),
  UPDATE_CONTROLS: () => {},
  UPDATE_DISPLAY: () => {},
  UPDATE_ENVIRONMENT: () => {},
  PLAY_CLIPS: () => {},
  UPDATE_ANIMATION: (state, speed) => (state.playbackSpeed = speed),
  UPDATE_LIGHTING: () => {},
  RESET_LIGHTING: () => {},
  UPDATE_ENCODING: () => {}
},

actions: {
  saveFileData({ commit }, fileData) {
    console.log('~~~ Saving file data to global state...')
    commit('SAVE_FILE_DATA', fileData)
  },
```

```
    resetFileData({ commit }) {
      console.log('~~~ Resetting file data to prepare for new model...')
      commit('RESET_FILE_DATA')
    },
    saveModelToHistory({ commit }, fileData) {
      console.log('~~~ Saving model to global state history...')
      console.dir(fileData)
      commit('SAVE_MODEL_TO_HISTORY', fileData)
    },
    saveModelInfo({ commit }, modelInfo) {
      console.log('~~~ Saving model info to global state...')
      commit('SAVE_MODEL_INFO', modelInfo)
    }
  },
})
```

Lighting.js

```
import { AmbientLight, DirectionalLight } from 'three'

/**
 * Creates lighting to then add to the Scene (default: Ambient and Directional)
 * with values taken from sceneState
 * @param {Array<THREE.Light>} lights Scene's lights
 * @param {Object} sceneState global scene state managed by VueX
 */
const createLighting = (lights, sceneState) => {
  // const lightHemi = new THREE.HemisphereLight();

  const lightAmbient = new AmbientLight(
    sceneState.ambientColor,
    sceneState.ambientIntensity
  )
  lightAmbient.name = 'ambient_light'

  const lightDirectional = new DirectionalLight(
    sceneState.directColor,
    sceneState.directIntensity
  )
  lightDirectional.position.set(0.5, 0, 0.866) // appx. 60 degrees
  lightDirectional.name = 'directional_light'

  return [lightAmbient, lightDirectional]
}

/**
 * Applies sceneState's lighting values to ambient and directonal lights
 * @param {Array<THREE.Light>} lights Scene's lights
 * @param {Object} sceneState global scene state managed by VueX
 */
```

```
const applyLighting = (lights, sceneState) => {
  lights[0].color.setHex(sceneState.ambientColor) // ambient
  lights[0].intensity = sceneState.ambientIntensity
  lights[1].color.setHex(sceneState.directColor) // directional
  lights[1].intensity = sceneState.directIntensity
}

/**
 * Removes all of Scene's lights
 * @param {Array<THREE.Light>} lights Scene's lights
 */
const removeLighting = lights => {
  lights.forEach(light => light.parent.remove(light))
  lights.length = 0
}

/**
 * Updates animation-related data, is called in Scene when notified by GUI
 * @param {Object} data object passed from Scene
 * @param {Object} sceneState global scene state managed by VueX
 */
const updateLighting = (data, sceneState) => {
  const lights = data.lights

  // Lighting control
  if (!sceneState.disableLighting && !lights.length) {
    const createdLights = createLighting(lights, sceneState)
    data.defaultCamera.add(...createdLights)
    lights.push(...createdLights)
  } else if (sceneState.disableLighting && lights.length) {
    removeLighting(lights)
  }

  // Exposure
  data.renderer.toneMappingExposure = sceneState.exposure

  // Default lighting
  if (lights.length === 2) {
    applyLighting(lights, sceneState)
  }
}

export default updateLighting
```

ДОДАТОК В

Веб-система для інтерактивної роботи з 3D об’єктами

Опис програми

УКР.НТУУ "КПІ". ТІ6294_20Б

Аркушів 10

Київ 2020

АНОТАЦІЯ

Додаток містить опис основних компонентів веб-системи для інтерактивної роботи з 3D об’єктами, а саме:

— Графічного інтерфейс користувача для взаємодії з завантаженою моделлю та 3D оточенням;

— Створення 3D сцени для графічного відображення об’єктів;

— Зберігання даних за допомогою централізованої схеми управління даними для всіх компонентів програми Vuex;

— Компонент створення світла у 3D сцені та управління ним, як приклад одного з пунктів меню графічного інтерфейсу користувача для маніпулювання 3D оточенням.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис основних компонентів веб-системи для інтерактивної роботи з 3D об’єктами. У додатку Б міститься програмний код відповідних компонентів.

Веб-система розроблена за допомогою мови програмування JavaScript, фронт-енд фреймворку для створення інтерфейсів користувача Vue.js та бібліотеки для відображення анімованої 3D графіки Three.js.

Кросплатформений мобільний додаток розроблений за допомогою мови програмування Dart з використанням інструментальних засобів для створення інтерфейсів користувача Flutter. Додаток потребує операційну систему iOS версії 8.0 і вище або Android версії 4.0.0 і вище.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Система надає функціонал для інтерактивної роботи з 3D об’єктами, а саме:

— Графічний інтерфейс користувача для взаємодії з завантаженою моделлю та 3D оточенням;

— Адаптивний завантажувач моделі у вигляді як поодиноких файлів, так і папок із багатьма файлами та ресурсами;

— Створення 3D сцени для графічного відображення об’єктів;

— Зберігання даних за допомогою централізованої схеми управління даними для всіх компонентів програми Vuex;

— Маніпулювання 3D моделлю та сценою у цілому, що можна декомпозувати на:

- управління світлом;
- автовідтворення камери;
- зміна кодування текстур;
- управління анімаціями;
- зміна навколишнього 3D оточення;
- додавання допоміжних інструментів до 3D сцени.

— Навчальний посібник та довідка ключових прив’язок апаратного контролю користувача.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Згідно з архітектурою, програмний комплекс поділяється на два модулі:

— Веб-система для інтерактивної роботи з 3D об’єктами (основний);

— Кросплатформений мобільний застосунок для класифікації найбільш поширених на даний момент типів 3D файлів (додатковий).

Веб-система поділяється на міксовані компоненти інтерфейсу користувача, що водночас слугують для відображення, поверхневої логіки і взаємодії з користувачем, та суто логічні компоненти та моделі для створення 3D системи, обробки поточного стану даних тощо.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для використання веб-системи користувач повинен мати персональний комп’ютер або мобільний пристрій з браузером, що підтримує JavaScript, а також підключення до мережі інтернет.

Для використання кросплатформеного мобільного додатку, користувач повинен мати мобільний пристрій з операційною системою iOS версії 8.0 і вище або Android версії 4.0.0 і вище. Мобільний пристрій також має мати підключення до мережі інтернет.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Веб-системний компонент програмного комплексу не потребує інсталяції завдяки використаній методології JAMStack, та доступний за посиланням https://penumbra3d.netlify.com.

Мобільний додаток потребує інсталяції, а інсталяційний файл можна згенерувати завантаживши необхідні файли за посиланням на систему керування та контрою версій GitHub https://github.com/oddestdan/penumbra_kb та виконуючи вказівки, описані у документації за згаданим посиланням. Згенерований файл необхідно запустити на мобільному пристрої, після чого буде проведена інсталяція. У разі успіху, додаток готовий до використання.

ВХІДНІ ДАНІ

Вхідна інформація веб-системи для інтерактивної роботи з 3D об’єктами:

— Інформація про використовуваний апаратний пристрій (браузер або операційна система у випадку мобільного застосунку);

— Завантажена 3D модель певного типу (яка успішно завантажується у випадку підтримуваного формату 3D файлу). Завантаження може відбуватися використовуючи як поодинокий файл, так і у вигляді папок із багатьма файлами та ресурсами.

ВИХІДНІ ДАНІ

Вихідна інформація веб-системи для інтерактивної роботи з 3D об’єктами:

— Інформація про завантажену 3D модель;

— Групи інтеракційної взаємодії з моделлю у вигляді графічного інтерфейсу користувача;

— Стисла інформація щодо ключових прив’язок апаратного контролю;

— Стисла інформація про розроблений проект та посилання на більш детальну інформацію.